Háskólinn á Akureyri
University of Akureyri

Viðskipta- og raunvísindadeild
Faculty of Business and Science

A comparison of game engines and languages

Final Year Project
2010

Rósa Dögg Jónsdóttir

Háskólinn á Akureyri
University of Akureyri

A comparison of game engines and languages

Final Report

Rósa Dögg Jónsdóttir

Supervisor: Andy Brooks

Computer Science Division
Department of Natural Resource Science

Faculty of Business and Science
University of Akureyri

Submitted April 2010, in partial fulfilment of
the conditions of the award of the degree BSc.

I hereby declare that this final report is all my own work,
except as indicated in the text:

Signature ______________________

Date _____/_____/_____

Abstract

This project compares three different game engines; Microsoft's XNA, Panda3D and Adobe Flash. The same game is created in all three engines and the programing experience and results compared. The strength and weaknesses of the integrated development environments (IDEs), length of code, run time information, and available support of each engine are among the variables compared.

Table of Contents

Illustration Index

Index of Tables

1 Introduction

Computer games are an increasingly popular past time and creating such a game can be a challenging and fun experience. There are a lot of choices of tools to create games available, both free and commercial (DevMaster.net 2009). Normally, choosing the tools is not the first thing a game creator does. First a game is designed, from basic idea to fully outlined storyboards. How ever, for this project the goal is to compare the game engines chosen, not decide the quality of the game. The main effort will therefor be put into the evaluation of game engines. A preexisting game is selected to expand on, rather than create an entirely new one. The same game will be implemented in all the engines to use as a comparison between the engines.

1.1 Game engine

A set of software tools to create a game is generally referred to as a game engine (Ward, 2008). The tools can be anything from an integrated development environment (IDE) for a programming language, with some basic code for starting and ending a program, to a highly developed software package with complete physics, rendering, network and artificial intelligence. The types of engines can be categorized according to difficulty in using them, platforms the game is for or by what type of games can be produced through them. Ward (2008) sorts game engines into three categories: Roll-your-own game engines (lowest level), Mostly-ready game engines (mid level), and Point-and-click engines (highest level). The lowest level depend the most on knowledge of the developers and the highest requires little to no expert knowledge.

This project assumes the creator has some basic programming knowledge and little to no money or professional experience. It also assumes the creator wants to practice programming and design skills. This eliminates any commercial product that does not offer a trial long enough to create a game in (for this project, that time is one month of implementation), any purely point and click tool where no programming is involved and tools that require expert knowledge to use. The limited available resources dictate that the game platform needs to be a platform previously accessible to the creator, so a desktop computer is the obvious choice.

The final choice were Panda3D, Microsoft's XNA and Macromedia's Flash. The engines were chosen because of different languages and availability. The engines for this project can all be categorized as 'Roll-your-own'. Panda 3D is a python game engine that can use either OpenGL or DirectX as a graphic engine. It is open source and has several available IDE's. XNA game studio can use either C# or C++ (and can convert one into the other). It was created by Microsoft and the recommended IDE for it is Microsoft Visual studio. C# was chosen because it is many ways simpler than C++. Flash is a popular web application format, it is written with ActionScript. Flash requires software from Macromedia that is not free but is available for 1 month trial. This way Panda 3D represents the general open source choice, XNA represents the free but corporate choice and Flash represents the costly corporate choice. Among other choices that were examined were SCALA, Slag, Java, JavaScript and Wild Pockets. Java was not chosen because the game had already been programmed in Java and Java is known to have huge memory consumption, a disadvantage against the other languages (Prechelt 2000a). Scala, Slag and JavaScript were not chosen because of their similarity with Java and C# in code and Panda3D in availability. Wild Pockets is a mostly point-and-click engine, though it is interesting because it offers 3D in web browser. The choices and evaluation of game engines is discussed further in section 2.1

1.2 The game

It could be interesting to design a game just for this project completely from start, but the time is limited and the goal is to compare engines not to create a whole new story. To have more time to focus on the comparison an existing game, old text game called Mugwumps, is chosen. Mugwumps is part of a series of games originally written in BASIC (Ahl, 1978) and was released in a bundle of Java games on KIDware (Java games, 2009). This game was chosen because it involves both chance and choices for the player, giving many possibilities for expanding the game play. The name also has a 'cute' factor. The other games in the Java games from KIDware had either complete randomness or little to no, leaving Mugwump as the game that balanced the two best.

The original text game is simple; Player is given a hint to how far the Mugwumps are from him and then guesses the Mugwumps location. To make the game more interesting for testing the engines it will be re-designed for graphics and the game play enhanced with a non-player character (NPC) and different difficulty of puzzles. The same game designs will be used for all game engines so that the comparison is as homogeneous as possible. Another more complicated game could have been chosen over Mugwumps. The design phase would then go into reverse engineering and there is no guarantee that would take any less time than designing a different game play for Mugwumps.

To have a clear idea of what path the game should take a target audience was chosen. It was decided to target young children, but that it might be enjoyable for older audience. Further discussion on the choice of game and game play is in section 2.2 and 3.

1.3 Comparison

There is a considerable amount of comparison made between programming languages, tools and game engines. Most of it is in opinion pieces and professional articles. Not many are empirical. Most of those that are empirical compare a limited amount of languages based on a relatively simple program. This can be attributed to the amount of effort involved in creating a complex program in multiple languages for the purpose of comparison. There are comparisons that use complex programs but they use different programs for different languages. Prechelt (2000a) points out how this provides an inaccurate sense of comparison. This project is not unlike those that have a limited amount of languages and a relatively simple program. It will give a perspective in what each engine can and cannot do but will not be a finite declaration of what is best. It is still relevant for someone who is considering what sort of engine to choose and the difference between these engines.

To compare the three engines a few factors will be looked into and evaluated. The comparison will be in part based on numbers and in part based on nominal attributes which will be assigned by the writer's experience.

- Code readability (nominal)
  - Readability is important if the program is to be extended, maintained or otherwise read after the creation.
  - It will be highly dependent on the programmer's skill but should be measurable between engines because the same programmer, and there for the same skill level, is applied on all.
- Lines of Code (numerical)
  - How long it takes to program a software is linked to how long the code is, but an engine is supposed to make the amount of programmer written code needed less. It will be interesting to see if the total lines of codes are significantly more than the programmer's written code.
- Implementation time (numerical)
  - How much time goes into implementing the game is important when considering what engine to use. A shorter amount of time would be better than longer.

- Functionality (nominal, list)
    - The functionality offered by the engines is interesting. It should make implementation time shorter the more functionality it offers. It can also make learning to use the engine more difficult if there are a lot of extra functionality
- IDE qualities (nominal, list)
    - The good points of each engines IDE.
- IDE weaknesses (nominal, list)
    - The bad points of each engines IDE.
- Official support (nominal)
    - How good the official support is, what support is available and how is it accessed.
- Community support (nominal)
    - How good the community support is, what support is available and how is it accessed.
- File size (numerical)
    - How large is the final product in KB. Has impact on accessibility of the final program.
- Run time memory consumption (numerical)
    - How much memory does the program consume on average. Has impact on usability.
- Load time (numerical)
    - How long does the program take to load. Has impact on enjoyment of play.
- Run time errors and glitches (nominal, list)
    - Amount of errors are arguably connected to the programmer's skill but because the same programmer (and there for the same skill level) is applied then the difference in error count and glitches (such as hesitation during play due to loading) should be dependent on the game engine.

Part of the choice of engines was to make sure it would run on specific platforms (a desktop) so compatibility will not be a measurement.

1.3.1 Engine

A game engine will provide a set of commands to make creating a game easier, this will be in addition what a 'clean' batch of code will have. What, if anything, is supplied by one engine and not another will there for be one step of the comparison. The game engines have different IDE's which will offer different usability (error checking, syntax checking) and access to documentation. This will be the second step to compare the engines. The third step will be to compare the apparent online community help, how easy or difficult it is to find solution or pre-made packages for common components. A distinction between help available for general coding in the programming language and specific for the engine will be made based on where this help is found.

1.3.2 Programming language

Chen et al (2005) divided programming language's factors into two categories; intrinsic and extrinsic. Intrinsic were traditional factors such as generality, reliability and efficiency and extrinsic were support of different groups. Prechelt (2000a) compared seven languages based on what would then be intrinsic factors such as; program length, readability, memory consumption and robustness. For this project the most interesting metrics would be memory consumption, how large the game files are, program length, loading time and readability. Memory consumption and file size are interesting for the distribution and use of the game while program length and readability are important to know how long it would take to develop and maintain a game. Further discussion on comparison is in chapter 2.3.

2 Research

The main scope of this project, and there for the research behind it, can be divided in three parts; the engine, the game and the comparison. Relating to the engine is programming languages and to the game is design.

2.1 Game Engine

There are over 3.000 programming languages in existence today (Kosar et al. 2008). Many are of the same dialect (Zilbert, 2001) and only 150 are considered popular enough to keep track of their use (TIOBE 2009). Any programming language can technically be used to create a game. For the purpose of this project, there needs to be an already existing game engine for the language. This engine may not require expert knowledge so it needs to have some fundamental rules for a game such as graphics management and game loop. The language to choose has to be fairly popular and easy to learn and use. This is so the creator can reasonably learn enough to create a game in a short amount of time. Chen et al. (2005) predicted that Java would be the most popular language in 2008 and according to TIOBE (2009) it shares the top seat with C in November 2009. Despite that, neither Java nor C will be chosen. The game has already been implemented in a text based version in Java and C variants, C++ and C#, are easier to learn than C and are relatively popular. C# with it's memory abstraction shortens the development time since the programmer does not need to worry about memory allocation (Carlson 2005).

The first engine chosen will use a C variant. Microsoft's XNA offers the use of either C++ or C#. XNA is a fairly new engine, released in 2007 and boasts a wide variety of online content for creators. XNA is used to create games for Windows personal computers, Xbox or Sune. C# will be chosen over C++ mainly because of C# memory abstraction. Engines that are similar to Microsoft's XNA would be SCALA and Slag for their language's similarity to C and Java (SCALA 2009, Slag 2009).

The C family of languages is often referred to as 'conventional programming language' (Prechelt 2000b). Conventional programming languages are distinguished from so called 'scripting' languages. Scripting languages are interpreted, rather than compiled, and are 'weak typed' (Prechelt 2000b, Carlson 2005). Scripting languages are heavier on memory than traditional because they are interpreted but quicker to write in because the code is usually shorter (Prechelt 2000a, Carlson 2005). Therefor a game engine that uses a scripting language offers a good contrast to the more conventional XNA.

Panda3D is an open source game engine that uses the Python programming language. Python is usually classified as scripting language, though it has some qualities of a traditional language (Carlson 2005). Panda3D runs equally on Linux, Mac-OS and Windows. IDE for Panda3D are many, from simple notepad to Eclipse. This is another distinct difference between the two engines. Similar options to Panda3D include Ruby and JavaScript (Ruby 2009, Wikipedia 2009). Neither Ruby nor JavaScript are nearly as popular as Python (TIOBE 2009). Ruby is new and shiny so might not have had the time to build up a knowledge base which would put it at a disadvantage to C#. JavaScript is much older but tied in with web browsers rather than stand on it's own.

Flash is the last choice. A popular way of creating games for the internet. It does not come free, unlike the other two. Adobe Flash CS4 Professional(c) costs 699$ in December 2009 (Adobe 2009). It does offer a 30 day trial which will be used, putting an extra time constrain on the implementation process. Why choose an expensive way instead of choosing something free? Because Flash is extremely popular as a web application and because it offers a comparison to the free choices. Are you paying for a better work environment or just for the technology. Flash can also be used to created games for other platforms such as smart phones. Flash uses ActionScript, it's own scripting language. Another option for a web.2 application would be Microsoft's Silverlight. Since there is already one engine from Microsoft, Silverlight is not chosen. It should be noted that Silverlight has the advantage over Adobe's Flash in that it is free.

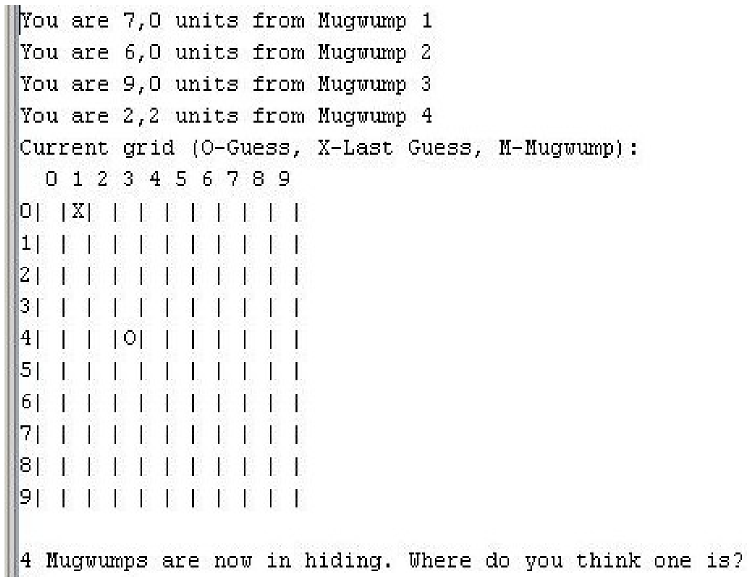
For a large software project it is not unusual to use more than one language to develop in. Scripting language are then used to 'glue' together components (Ousterhout 1998). There are several game engines available that use more than one language, but in the interest of keeping this project simple they will not be considered.

2.2 The game

2.2.1 Origin

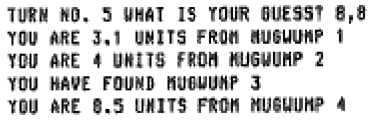
In the Java games bundle from KIDware, in addition to Mugwumps, there were: Acey-Deucy, a card game; Even Wins, a game where the user and computer take turns removing markers; Lunar lander, where the user lands a spaceship using considerable amount of math to succeed; Frown, a random dice game; and Jot, a word guessing game. These games either rely heavily on chance or knowledge of the user.

None of them struck as a game that would be 'fun' to develop further with the creation of NPCs and variability in puzzles. Mostly it felt as if they lacked 'story' (except Lunar lander). Looking for Mugwumps sounded like it had a story behind it that wasn't being told. The word Mugwumps is derived from an Algonquian dialect of Native Americans and means 'war leader'. The word has since changed meanings several times in use in English, from meaning 'turn coat' to representing a wizard's title in a popular children book series (Quinion, 2007).



```
You are 7,0 units from Mugwump 1
You are 6,0 units from Mugwump 2
You are 9,0 units from Mugwump 3
You are 2,2 units from Mugwump 4
Current grid (O-Guess, X-Last Guess, M-Mugwump):
  0 1 2 3 4 5 6 7 8 9
0| |X| | | | | | | | |
1| | | | | | | | | | |
2| | | | | | | | | | |
3| | | | | | | | | | |
4| | | |O| | | | | | |
5| | | | | | | | | | |
6| | | | | | | | | | |
7| | | | | | | | | | |
8| | | | | | | | | | |
9| | | | | | | | | | |

4 Mugwumps are now in hiding. Where do you think one is?
```

Illustration 1: Java game - Screenshot



```
TURN NO. 5 WHAT IS YOUR GUESS? 8,8
YOU ARE 3.1 UNITS FROM MUGWUMP 1
YOU ARE 4 UNITS FROM MUGWUMP 2
YOU HAVE FOUND MUGWUMP 3
YOU ARE 8.5 UNITS FROM MUGWUMP 4
```

Illustration 2: BASIC game - Screenshot

The original game and the Java game are not completely the same (see Illustration 1 and 2). Mugwumps is a simple 'hide and seek' game. The mugwumps are hidden in a grid and the player is given hints to where they are. The player has to try to find the mugwumps in as few guesses as he can. The original version of Mugwumps is a text based game with no graphical representation of the grid the mugwumps are hidden in, it states how far the Mugwumps are from the player's location and how many tries the player has left. In the original BASIC game the player had limited tries but in the Java version the player has unlimited tries and the grid is represented using | and _ with marks for where the player has guessed and failed and where the Mugwumps that have been found are. Such small changes make the game considerably easier.

2.2.2 Design

Blow (2004) claims 'Making a game today is a very different experience than it was even in 1994.' This is because a game has gone from being in 2D with basic I/O streaming and some simulation into containing 3D rendering, AI, various scripting, physics and more complex worlds and story lines. That is true in part, yes there are more complicated games available now but it can be argued that the variability in complication is wider rather than all games are more complicated. Game design will have more sub stages for more complicated games but it could be said that the basic steps are timeless. Crawford (1997) describes the stages to be: Choose a goal and a topic, research and prepare, design (starting with the game, ending with program), programming, play testing, and finally postmortem. Ab.Rahman & Prakash (2007) wrote, on creating a game in XNA, that '[...] the most important matter that should be highlighted is the content team members' perspectives. '. This is similar to Crawford's emphases on choosing a goal and topic first, having a clear idea what the game should be before going into programming.

Hunicke et al. (2004) suggest that: 'MDA [Mechanics, Dynamics, and Aesthetics] is a formal approach to understanding games, one which attempts to bridge the gap between game design and development, game criticism, and technical game research. ' Using the MDA model a designer can tune game mechanics to meet player's expectations of aesthetics. This is done by mapping aesthetics (game play component such as sensation, discovery or narrative) to specific dynamics (systems such as time) and specific dynamics to certain mechanics (rules such as spawn points and available items). See Illustration 3.

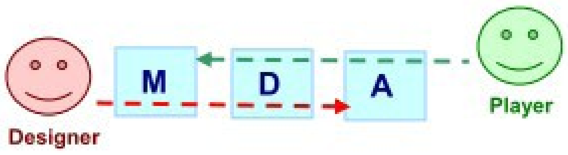


Illustration 3: MDA (Hunicke et al 2004)

Knowing who the player is is a step in knowing how to approach the player. There are several different ways to classify a target audience for a game. Broadly they are marketed for 'boys', 'girls' or 'gamers'. Games for 'boys' involve some variation of sports or fighting enemies. Games for 'girls' involve 'cute' puzzles, dressing up or anything pink. Gamers are a large and diverse audience but the games are advertised for one or more of: a type of action, storyline, graphics – depending on what type of game it is. Most target a male audience. There is a growing group of audience, so called casual gamers which mostly consists of women over 40 – according to an AOL study the biggest online gaming group (CNN 2004). This group is thought to be mostly into online Flash or Java games but is little researched. Mugwumps is a puzzle game so the appropriate group would either be children (or specifically girls) or casual gamers. Because the casual gamer is a very ambiguous audience children will be the targeted.

Games are an important factor in children lives and some consider them a part of teaching them social skills (Eggerstdóttir & Petrova 2008). Computer games have often been classified as less prosocial than other games. Children's exposure to violent video games result in increase in aggression and decrease in helping others (Gentile & Walsh 2001). There are numerous articles about violence and video games for children, almost as many about teaching through video games (math games, spelling games and such) but very few about positive effect of video games on children through play. Gentile et al. (2009) looks into the effects of prosocial games on children and finds that ' in the shortterm players' behaviors were predicted by the pro social and violent content of the games they played. In the long-term, players with high pro social game exposure had higher prosocial traits and behaviors.'. This means studies have shown that video games are not all bad, it depends on their content. Mugwumps as a hide and seek puzzle game should there for be relatively safe for children and maybe even helpful for reading skills.

Laurel (1998) talks of her experience in developing a series of games for girls. She mentions how she did an in-depth research into what girls would want to play – asking what they liked rather than 'of what we have already, what do you like best'. This project doesn't have time to do such a research so will rely on existing material but it is an interesting point.

Games for children usually have simple forms and colours. The graphics are figurative rather than realistic and physics are unreal – except when the game teaches physics. Good teaching games for children give the player a lot of compliments when a correct action is performed (Sigurðardóttir 2007). Wight (2003), creator of SIMs, talks about how to make games interesting by making failures interesting – not just successes. He also talks about the importance of human and computer interaction and that each player might experience the game in a different way. He is referring to more complex games than what Mugwump will be but it is still something to keep in mind. That if a player does something unexpected it might be an idea to not punish it or display an error but instead do something surprising.

2.3 Comparison

Prechelt (2000a) did an empirical comparison of seven programming languages. Languages were divided into two groups; scripting (Perl, Tcl, Python and Rexx) and conventional (C, C++ and Java) programming languages. One set of requirements was used to be implemented in all languages, conventional by students and scripting by members of online communities. Each language had several different programmers. Program length, effort, run time efficiency, memory consumption and reliability among other things were measured, the data collected into box plots and compared. Because there were different programmers, with different skill set and experience, for each language the measurements were spread but there were a few statistics that were significantly different between languages. Namely that scripting took half as long writing as conventional languages (due to shorter code), consumed twice the memory (except Java, which was significantly more memory hungry than any other language) and took longer executing (though again Java was slower than C and C++ and sometimes slower than scripting). Prechelt points out that this is only valid for this specific problem and that different programs would potentially give different results. He 'observed no clear differences in program reliability among the language groups'. Prechelt maintains that 'The scripting languages, however, offer reasonable alternatives to C and C++, even for tasks that must handle fair amounts of computation and data.'. This project is in many ways similar to Prechelt's comparison although the scope is much smaller. It will be interesting to see if the program length, effort and memory consumption is similar to Prechelt's results. The program run will be different from Prechelt's, his resolves a sorting problem, and there for there might be differences.

Chen et al. (2005) did a study on programming language trends. They used 17 third-generation general-purpose languages, excluding other generations and scripting languages. They surveyed general design criteria and support of different programming languages in 1993, 1998 and 2003. Then used this data to predict how much the same languages will be used in 2008. The general design criteria were factors such as generality, reliability, simplicity and implement-ability. Each language was given a grade for each factor by the researchers. The support was divided into institutional, industrial, governmental, organizational, grassroots and technology. Each support factor was measured either with surveys or data mining. They 'found that the most important intrinsic factors are generality, reliability, machine independence, and extensibility.' This research did not assume one language affecting another popularity nor did it predict new languages and their influence. This makes the measurements and predictions slightly skewed. The study completely ignored scripting languages.

Carlson (2005) discusses other comparisons, essays and evaluations. Looks specifically at scripting languages and the lack of research into them. Looks into what defines scripting (interpreted) languages. Discusses the difference between scripting and non-scripting languages. Mentions macros. Touches on industry focus on conventional languages and discusses advantages in using scripting rather than conventional programming. He advices developers to look into what language to use with balance in mind between memory consumption and cost of development. He mentions 'pretty' as a valid comparison due to being linked to readability. His advice is arguable in that he does not account for cost in maintaining the program after release, readability might have influence in that cost but so would extendability. The cost of correcting errors in old or new code should also be taken into account. The immediate cost of development might appear smaller at the cost higher maintain cost.

Another form of comparison would be to compare the actual code, word for word, such as Chandra (2005) comparison of Java and C# and Miller & Ranum comparison of Python and Java. These are interesting from a purely typing in code perspective and learning one language if the other is already familiar. Those are relevant in predicting how easy or difficult it is to learn a language and help a programmer migrate from one to the other. It is not relevant to the comparison of the game engines.

3 Requirements Analysis and Design

Designing a graphic computer game requires not only traditional software specifications in UML and use cases but also storyboards and concept art. This part of the report discuses basic software requirements and design decisions for the storyline, interface and graphics.

Software requirements with use cases are listed in more detail in appendix C – Software Requirement Specification document. It lists all decisions and requirements for the game for all three implementations.

A prototype in XNA was created for the text based game, the main code from that and screen shot is in appendix E. It is mostly code adapted from the Java version of the game with some code to make it work with graphics. ScreenManager is a set of classes provided from the XNA creators club and is not provided in the appendix (XNA Creators Club Online 2007).

Testing and comparison design will be left for the evaluation period of the project.

3.1 Software requirements

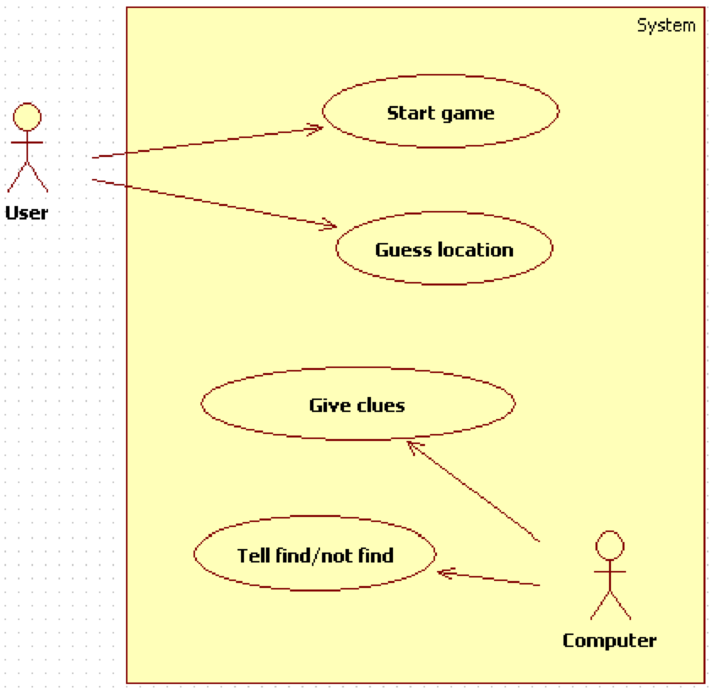


Illustration 4: Java game - Overview

Mugwumps is a simple 'hide and seek' game. Objects are hidden in a grid and the player is given hints to where they are. The player has to try to find the objects in as few guesses as he can. The original version of Mugwumps is a text based game with no graphical representation of the grid the mugwumps are hidden in, it states how far the Mugwumps are from the player's location and how many tries the player has left. See Illustration 4.

The goal and idea behind the game is to expand the previous versions of Mugwumps with graphics and additional playability. There needs to be a guideline into what direction that is designed. To have a clearer idea of where the design should be headed the target audience 'children' is decided, as discussed in 2.2.2 Design. 'Children' is a broad range so a decision was made to limit it to ages 6 – 10. Children younger than 6 will probably not have the reading knowledge to accept hints in a text.

Children 10 and up will possibly consider a simple game too 'childish'. While the game will focus 6 – 10 there is nothing that prohibits a person of any age to use it.

The game will be divided into three levels (called chapters); one training, one moderate difficulty and one hard. For future development of the game more chapters can be added, but for this project three is enough because they will be implemented three times. Having more chapters would be too much work in the short time available. The user will use a mouse to point and click to where he thinks the hidden object is and will be given graphical hints to in what direction the object is hidden and how far he is away from it. Graphics will show the possible locations where things can be hidden and where the player has already searched. See Illustration 5.

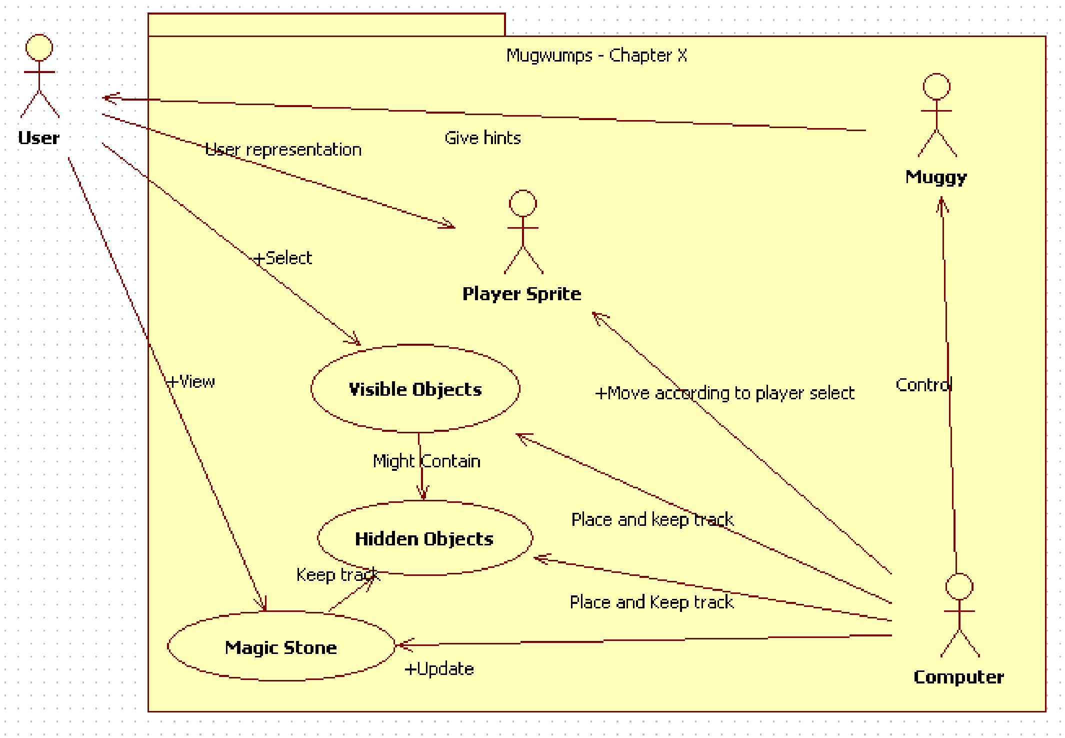


Illustration 5: New game - Overview

Creating a UML class diagram would be the logical next step for a software but because the game will be created in three different languages, C# only one of those that uses classes, a high level diagram of system components will do until implementation to give an idea of what each method or class will do. See Illustration 6.

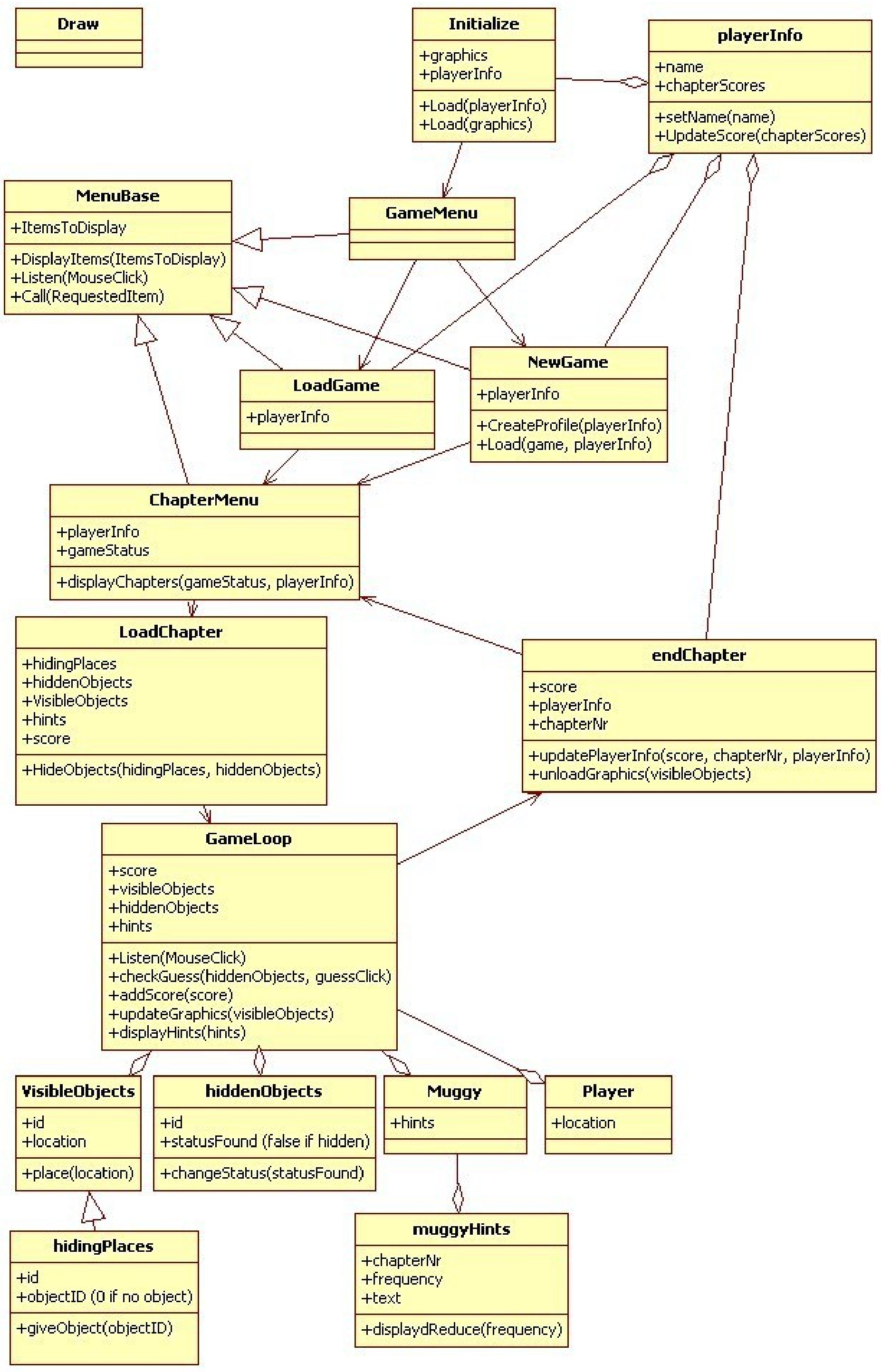


Illustration 6: New game - Overview II

3.2 Interface

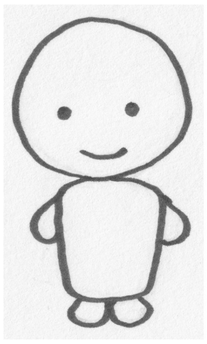


Illustration 7: Player Sprite

The game has to be cheerful, colourful and simple. Graphics will be in 2D on a 3D plane like paper dolls. Sounds will be unrecognizable squeels of joy or gasp to separate the game from a specific language. Text will be at a beginning reading level.

The figure of the Mugwumps will have a big head, ears and eyes like a puppy or a kitten because that is usually considered cute (see Illustration 8). The figure (sprite) of the player will be a really simple blob-person (see Illustration 7). For future development of the game the figure could have attachable accessories for the player to distinguish it from other players sprites.

Illustration 8: Muggy

The main game window will have the game board, Muggy's hints and the magic stone. Quit (X) and Help(?) need to be accessible there as well. The Magic Stone will display a coloured dot in the direction of where the hidden object(s) is. The colour will be brighter the closer the player is to the hidden object. Part of the game is to keep score. The number of guesses a player has done in the chapter will be displayed in the top of the screen. See Illustration 9.

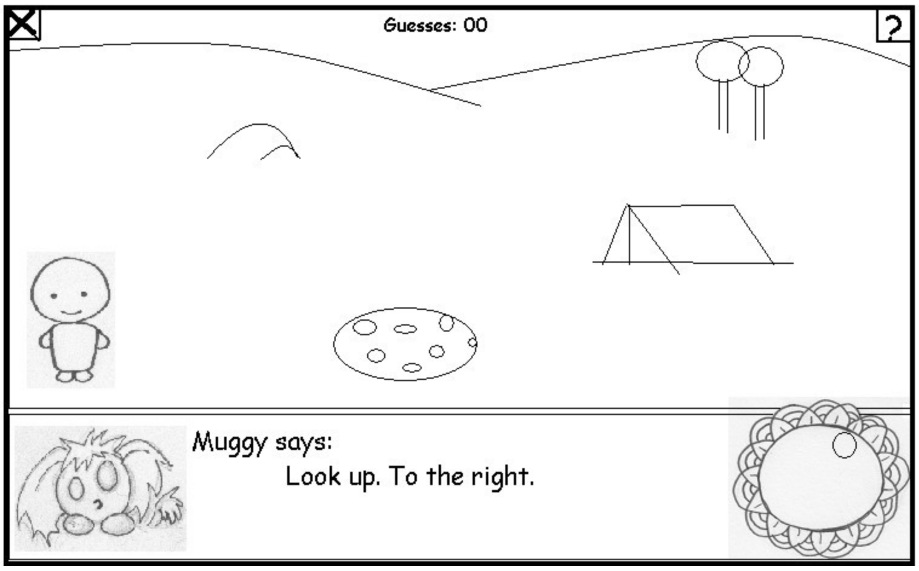


Illustration 9: Interface

Once score is kept the player might feel a need to do better. The computer has to keep track of the player's score so that he can see if he's doing better. So that more than one player can play using the same installation without interfering with the others score the computer needs to keep separate track of player information. This means that before the player starts playing the game he needs to be able to choose either to create a new game or load an existing one. See illustration 10 for Menu overview.

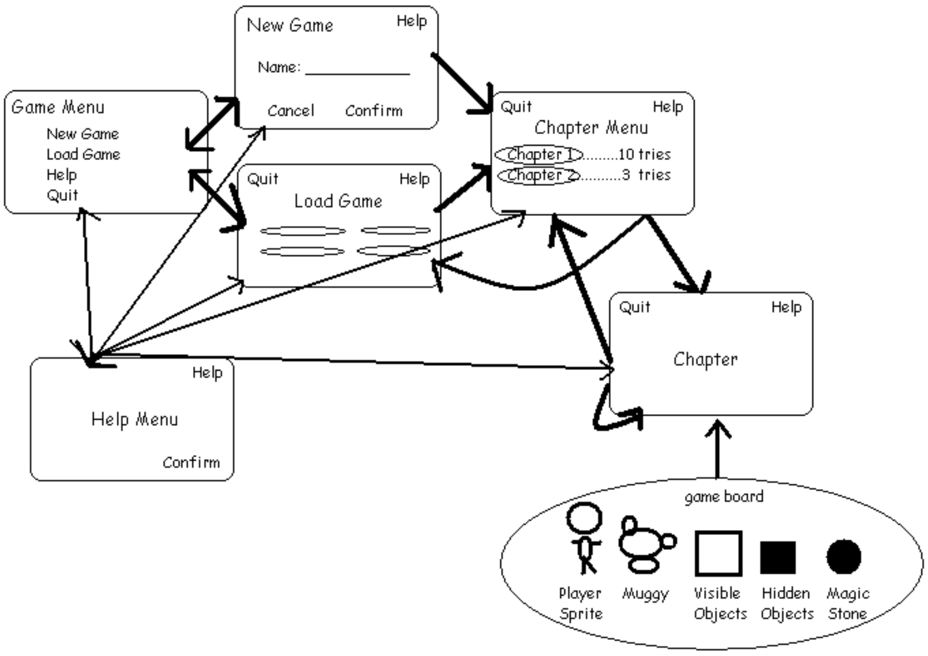


Illustration 10: Menu overview

An alternative to the Game Menu → New Game | Load Game would be to have one window for load game and new game (Illustration 11). This is not as clear as having two separate windows and might create confusion in new players.

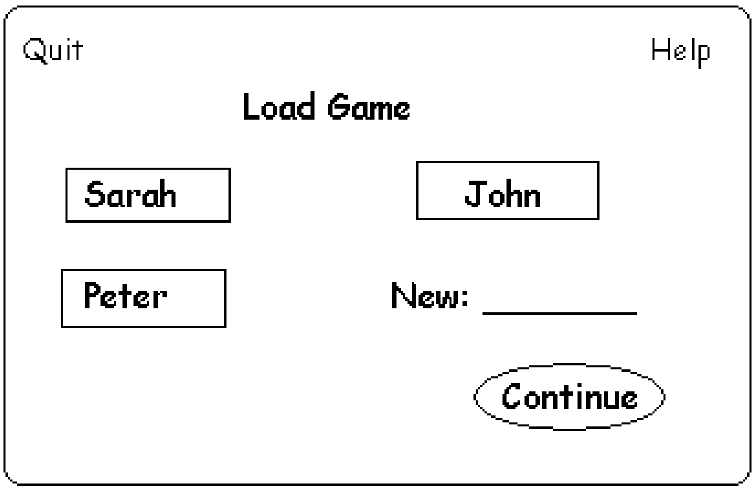


Illustration 11: Load game alternative

3.3 Storyline

The game starts with a creature (Muggy) finding the player (known as 'PC') and telling PC about 'games' the Mugwumps want to play. Muggy gives a magic stone that shows in what direction the hidden objects are. The story is three chapters, each a different level of the game. The first is a practice chapter, where the player needs to find Muggy in a field. The second a part of 'the games' where he needs to find three Mugwumps in a tent camp. The third chapter is a treasure hunt and more difficult than the others. Muggy is with the player to help and give hints.

In the original version the game flow was fairly simple (Illustration 12) – the player would receive hints and guess the location of Mugwumps until all Mugwumps were found.

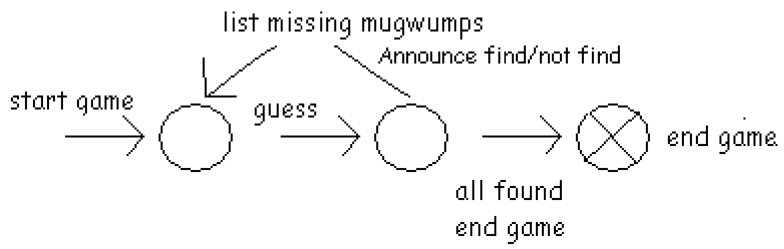


Illustration 12: Java game - Game flow

The basics of the flow will remain the same for the expanded game but with more 'actors'.

When the player selects a chapter to play the computer will initialize the game components. It will check for the hints available for Muggy, it will check for the visible and hidden objects and place them on the board. Finally it will display the game window complete with both interface and board. See Illustration 13.

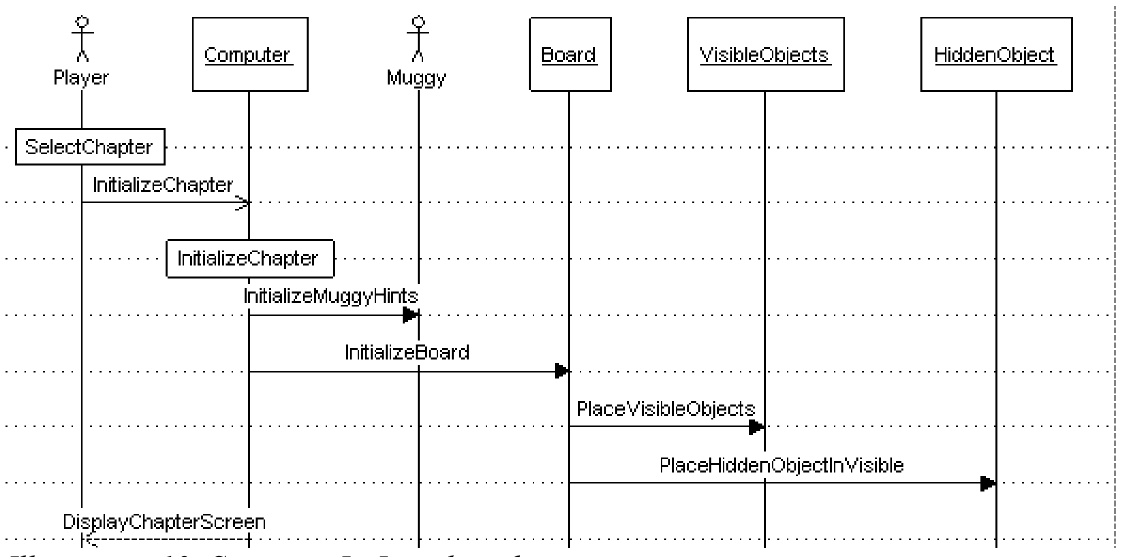


Illustration 13: Sequence I - Initialize chapter

At the beginning, when the player is idle, and any time he returns to idle state for more than 30 seconds, the magic stone displays a visible hint to where the hidden objects are and Muggy updates his hint to urge the player to find the hidden objects. See Illustration 14.

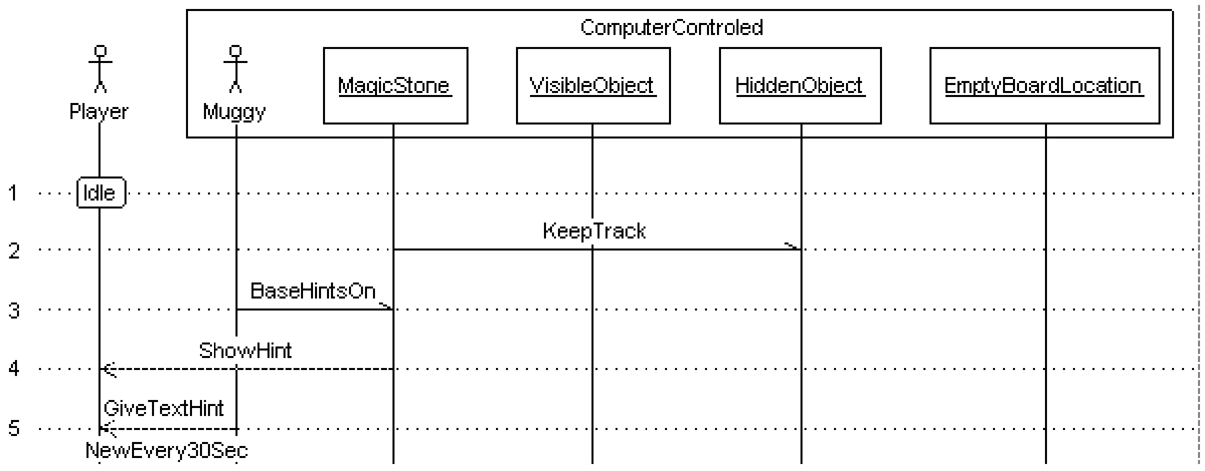


Illustration 14: Sequence II - Player Idle

When the player clicks the mouse, the computer will check where he clicked and the game will react in different ways. See Illustration 15.

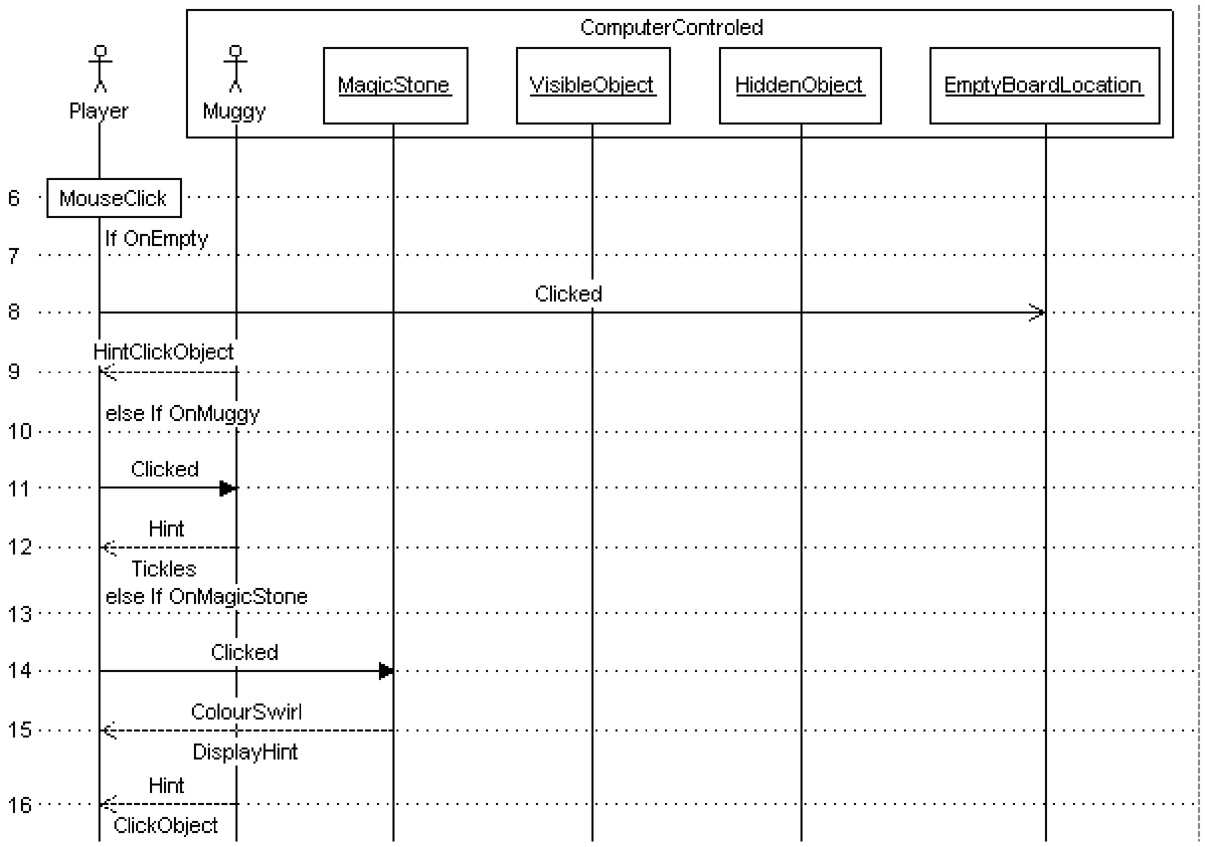


Illustration 15: Sequence IIIa - MouseClick I

If he clicked an empty spot Muggy will remind him to click an object. If he clicks Muggy or the Magic stone they will animate then remind him where the hidden objects are. See Illustration 16.

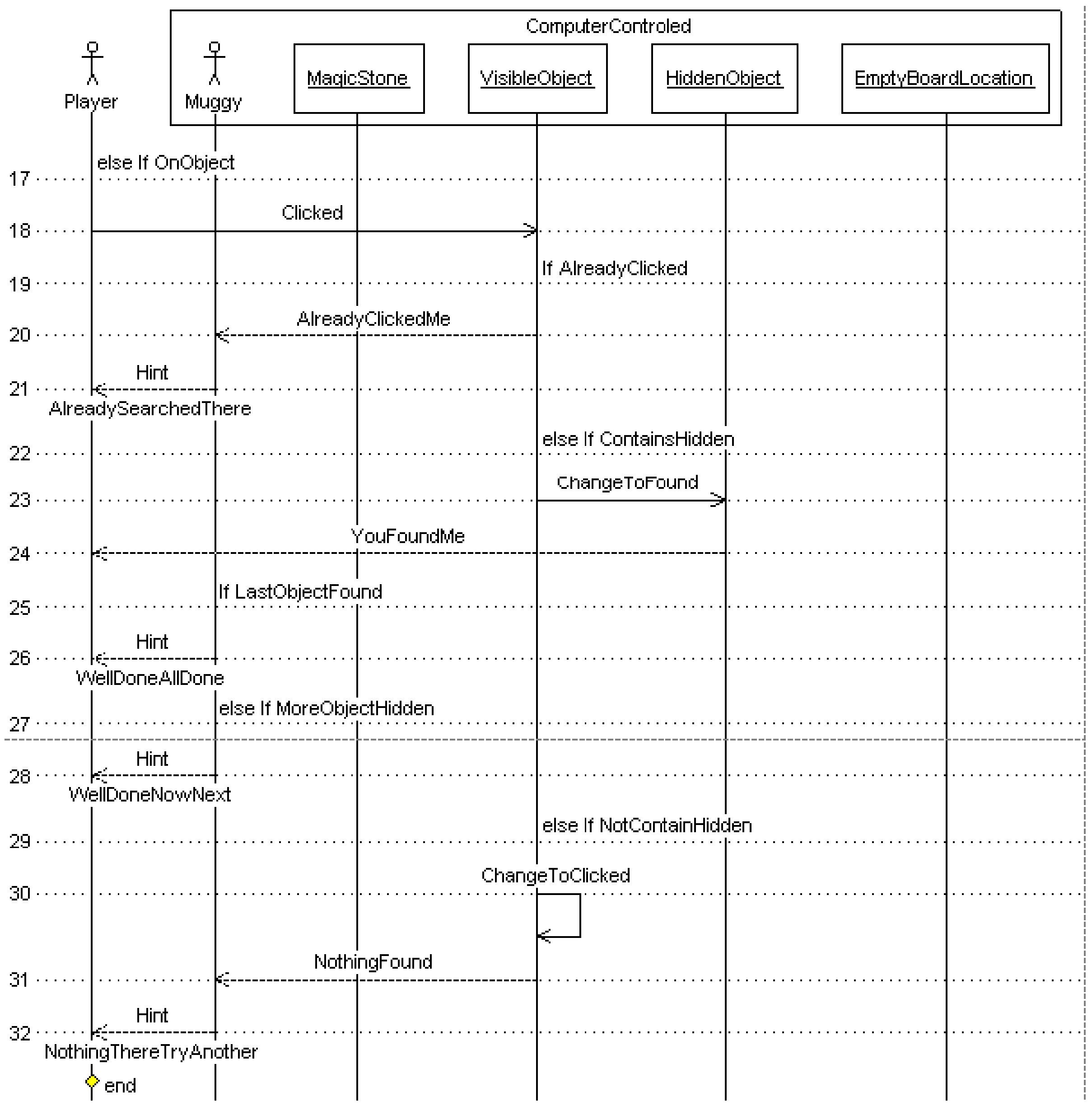


Illustration 16: Sequence IIIb - Mouseclick II

If the player clicked an object a series of checks are performed behind the scenes. If the object has been clicked before Muggy reminds the player and urges to look elsewhere. If the object has not been clicked and contains a hidden object, the hidden object becomes visible and Muggy congratulates the player. If this was the last object the game should end but if it was not then Muggy will urge the player to find the remaining object(s). If the visible object does not contain a hidden object then Muggy will urge the player to try something else.

The storyboard for chapter 1, the practice chapter where Muggy hides in a field and the player has to find him, is fairly simple (Illustration 17). Chapter 2 would be similar except that there are more objects. The stone then shows more than one dot and Muggy would say things like 'oh we're really close to one, try going right'. In chapter 3 there are no objects to hide behind, the field is clear. Then the player can click any area of the board and know if it is hiding the treasure chest. When an area has been clicked a hole should be displayed to represent the player digging for a treasure chest.

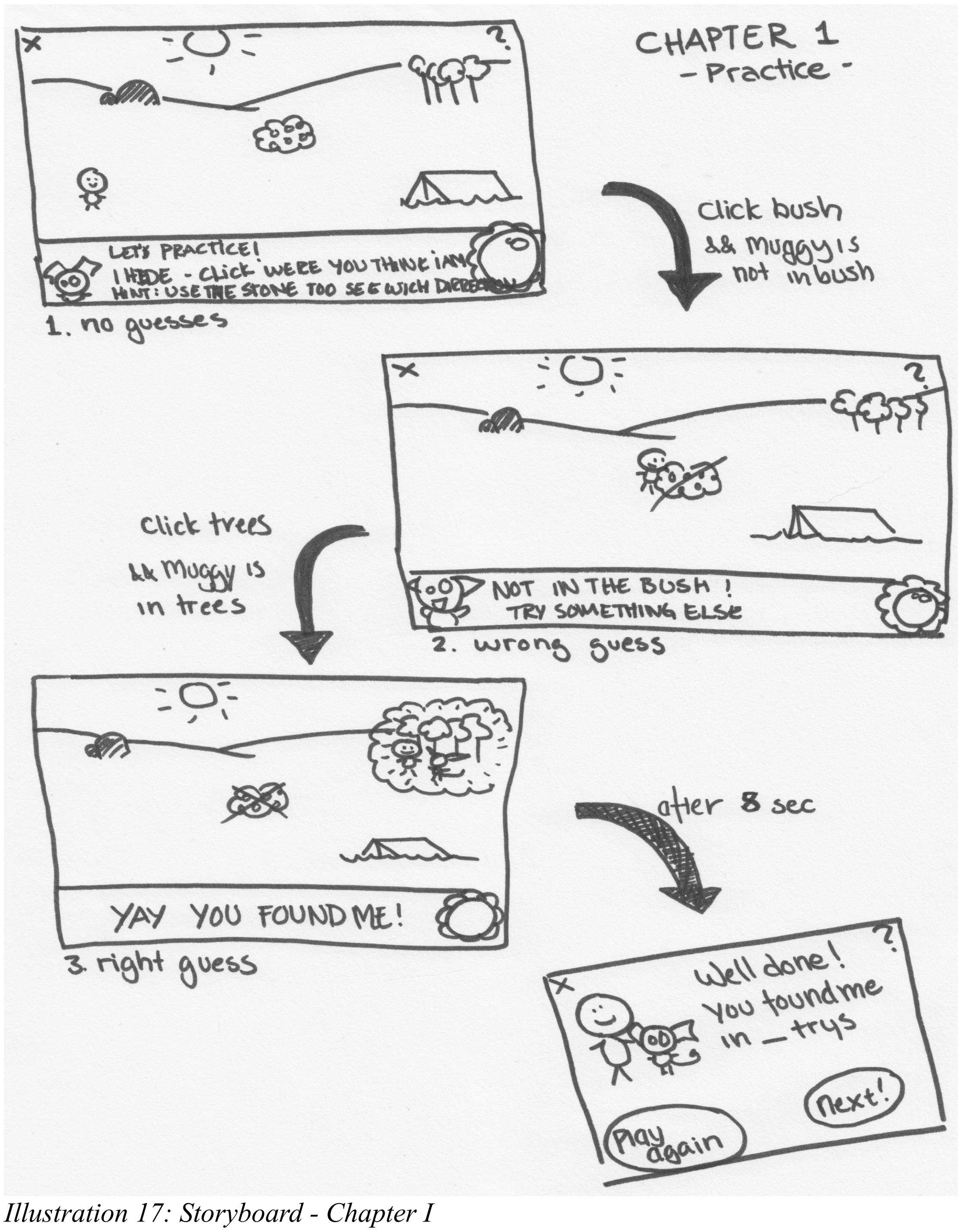

CHAPTER 1
- Practice -
LET'S PRACTICE!
I HIDE - CLICK WERE YOU THINK I AM
HINT: USE THE STONE TOO SEE WICH DIRECTION
1. no guesses
Click bush && Muggy is not in bush
NOT IN THE BUSH !
TRY SOMETHING ELSE
2. wrong guess
Click trees && Muggy is in trees
YAY YOU FOUND ME!
3. right guess
after 8 sec
Well done! You found me in _ trys
Play again
Next!

Illustration 17: Storyboard - Chapter I

4 Implementation

4.1 Technology Platform

All the work was done on a Dell Inspiron 6400 laptop with Windows XP, Service Pack 3 operating system. Open Office 3.2.0 and Notepad++ v.5.6.6 were used for notes and report making. MS paint and Gimp 2.6.8 were used to create simple 2D graphic and starUML used in making the use cases and flow charts. This could all be exchanged for a workstation and tools of the user's choice.

The important part is the game engines. For Microsoft's XNA, Microsoft Visual C# 2008 Express Edition was used. An alternative to that would be to buy Microsoft XNA Game Studio. XNA uses .Net and DivX, version 3.5 SP1 was used of .Net and version 9 of DivX For Panda3D, Eclipse SDK 3.5.1 with PyDev. Panda3D does not have an official IDE, instead they recommend a variety of IDE's on the Panda3D webpage. DivX was used for the graphic engine, but OpenGL can be used as well. Finally for Adobe Flash, Adobe Flash C4 Proffessional 30 day trial was used. There are free alternatives for Flash, some with graphic IDE and others purely text. To run the flash version Adobe Flash Player 10 is used.

For static analysis of XNA three extra programs were used. StyleCop and FxCop from Microsoft and SourceMonitor from Campwood Software.

4.2 Changes from Analysis and Design

There is some difference between the design and the final version of the game in each version. Mostly less functionality or features. The core functionality of the game was implemented in each engine and graphics and other features added as time allowed. There is a difference in the final versions between the game engines because of the different amount of time it takes to implement features in the engines. The XNA version is furthest along although it used up roughly the same amount of time as Panda3D. Panda3D graphics used up more time than expected. Flash version is close to Panda3D but the least time was given to that. Each game engine is addressed in the following sections, in what is different from the design and final version.

4.2.1 XNA

As mentioned, the XNA version is closest to the planned game of the three. It is the most object-oriented code, with special classes to handle object states and functions. The main difference between planned game and XNA version are as follows.

Less functionality

- Main Menu only accepts keyboard. Only displays Play Game, Options and Exit.
- No Save or Load game option
- No help menu
- Game Window accepts mouse clicks and ESC key. No Help. No Magic Stone.
- New game through ESC from game board.
- No magic stone to visualize the hint.
- No sounds.

The XNA version did look very close to the storyboard graphics (Illustration 18).

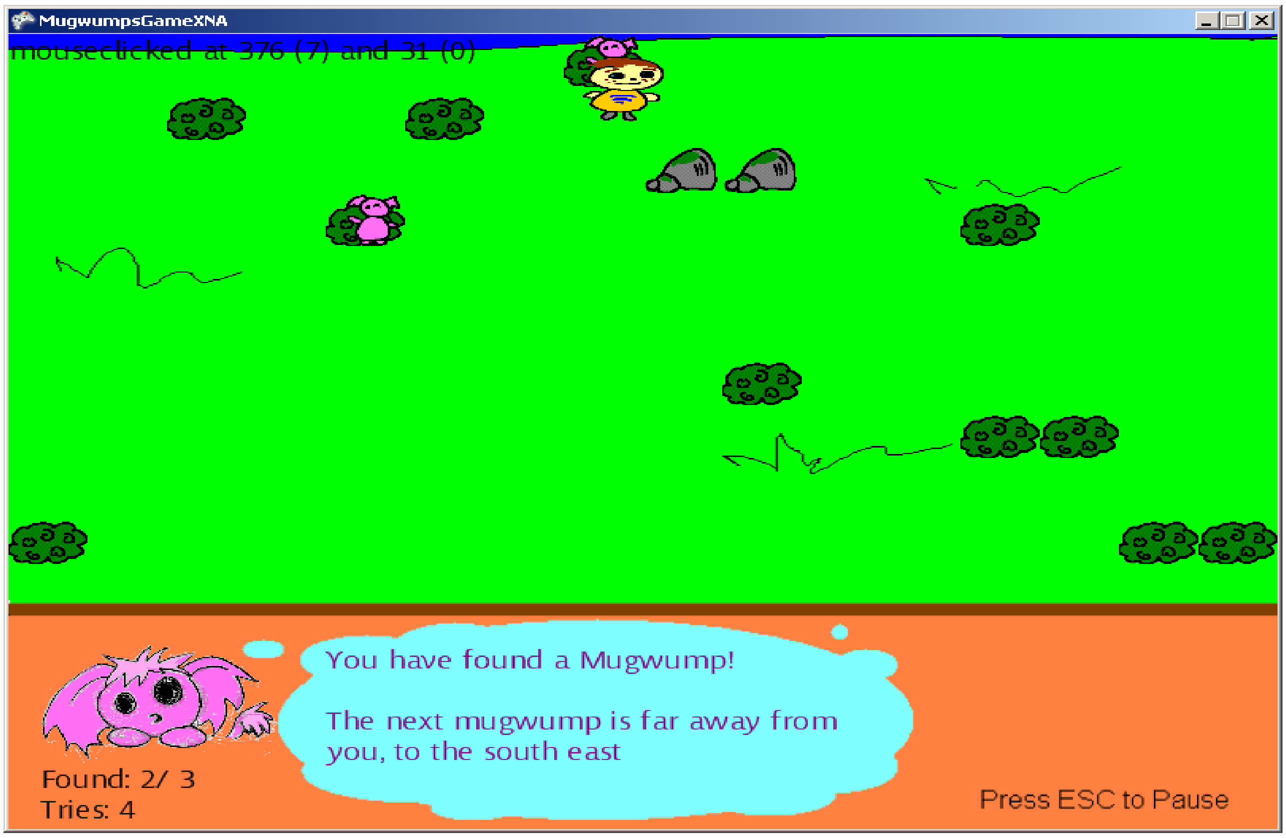

MugwumpsGameXNA
mouseclicked at 376 (7) and 31 (0)
You have found a Mugwump!
The next mugwump is far away from you, to the south east
Found: 2/ 3
Tries: 4
Press ESC to Pause

Illustration 18: Screenshot - XNA version

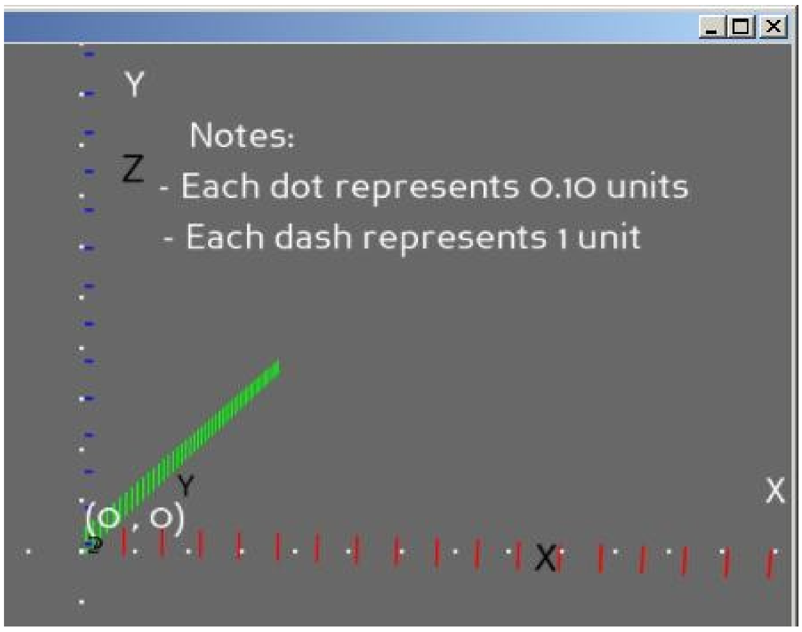
A screen manager was used from the XNA community to create the menus. Using code from others saves allot of time. As the refactoring tools were run it showed that the code that is borrowed is not always up to standards and it takes more time to correct someone elses code than ones own.

The decision to leave these out was made because it was time to focus on the next implementation (in Panda3D). It should be fairly easy to add menu options and help to the game. Saving and loading a game as well as sounds was not looked into in debth but the XNA engine should support both. Adding a magic stone to visualize the hint can be done by adding a new class.

Leaving these things out means that the game is not ready for release, but it has sufficient functionality to be a demo.

4.2.2 Panda3D

Graphics were a major issue in Panda3D. The final version of the game in Panda3D is in 3D while Flash and XNA are in 2D. This is because of how Panda3D handles coordinates differently than Flash and XNA. In all three engines, graphics can be in either 2D or 3D. Both 2D coordinations of the game window and mouse possitionon the screen for XNA and Flash are on the same axis. In Panda3D the mouse is a string from the nearest to furthest away camera angle, the game window has it's own x and y coordinates and the *scene* (3 dimensional box the game enviroment inhabits) has it's own x, y and z coordinates (Illustration 19).



Notes:
- Each dot represents 0.10 units
- Each dash represents 1 unit

Illustration 19: Screenshot - Panda3D Coordinates

Trying to place all of these coordination in synchronization took a long time and turned out unsuccesfull. Instead a chess tutorial provided with Panda3D was used as a game board and the pawns used as bushes. The basic logic behind the game is still the same but the look and 'feel' is different.

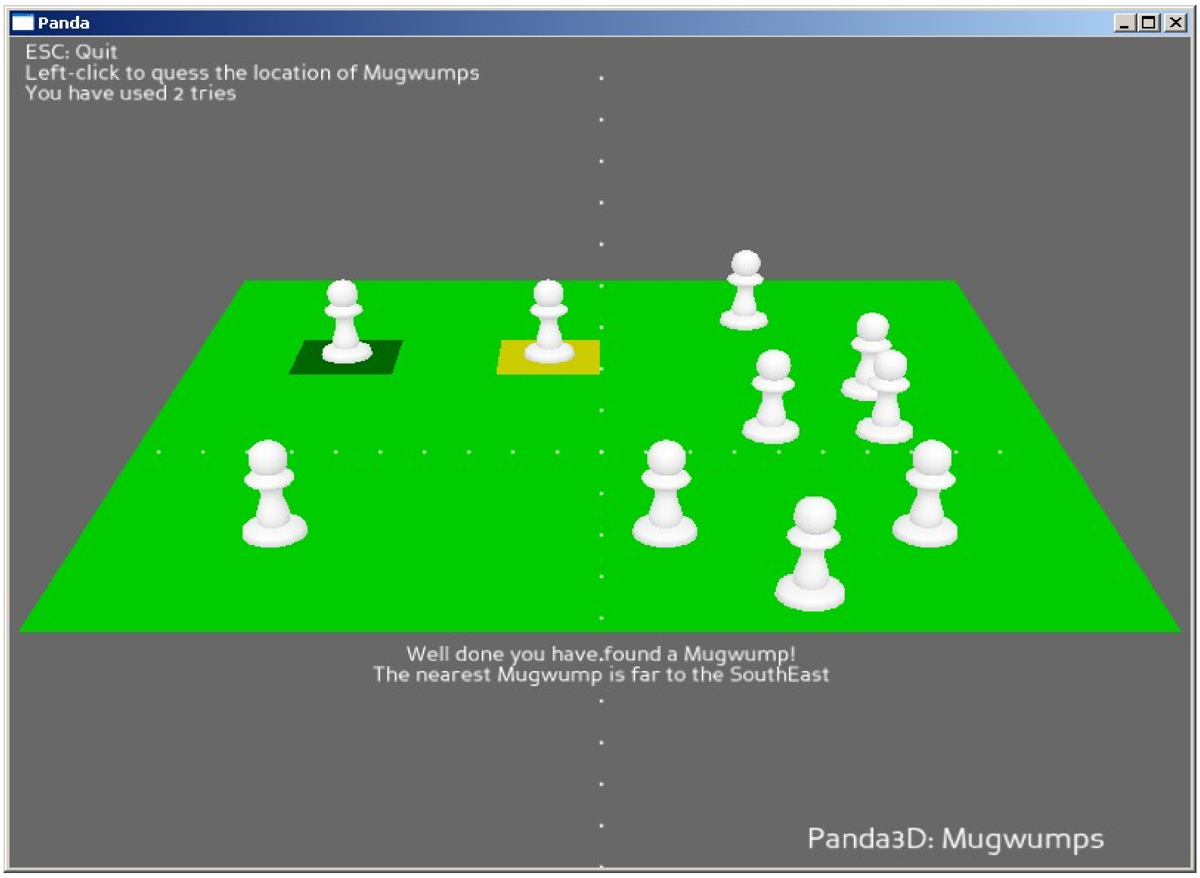
Less functionality

- The player sprite was not added due to running out of time.
- No help
- No main menu to select new game, no replay, no save or load
- No sound, no magic stone

Different gameplay

- 3D graphics
- No player sprite

Panda3D version looks different from the original storyboards, even though the layout is based on the same ideas. (Illustration 20).



ESC: Quit
Left-click to quess the location of Mugwumps
You have used 2 tries

Well done you have.found a Mugwump!
The nearest Mugwump is far to the SouthEast

Illustration 20: Screenshot - Panda3D version

Creating a menu window to be able to select options should not be a big problem but was not attempted due to time constraints. Neither was exploring how Panda3D handles saving and loading of game data.

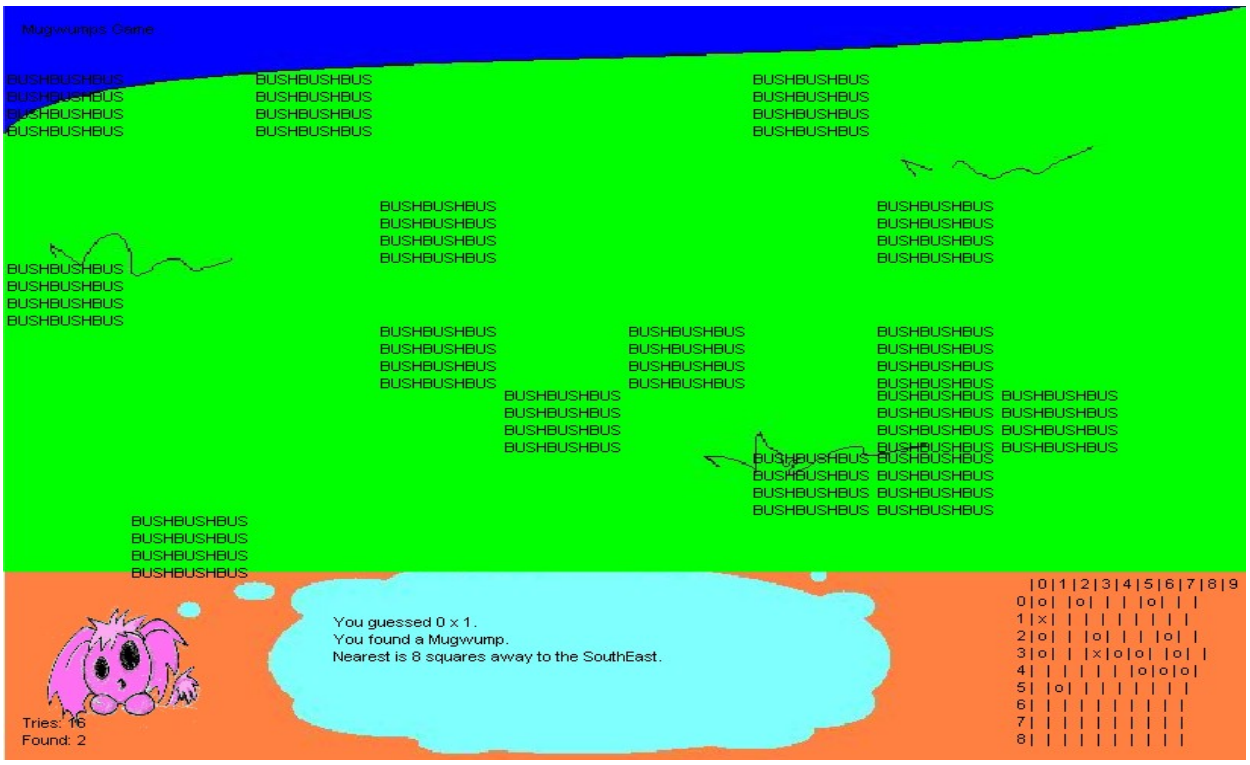
Because the graphics are pawns and the game does not look and feel like a hide and seek game this version is not a demo. Instead it could be called a prototype. It is playable to get a feel of what the game is about, but it does not demonstrate fully how a new player would experience a potential final game.

4.2.3 Flash

The main disadvantage of Adobe Flash IDE was that it was only a trial that was active for 30 days, limiting the amount of time to implement and test everything. It is also the most graphical of the IDEs. Graphics are drawn, using mouse point and click, on the scene and then a script is attached that handles the game logic. This makes working with Flash allot different than the text focused IDEs of Panda3D and XNA.

Less functionality

- The player sprite was not added due to running out of time.
- No help
- No main menu to select new game, no replay, no save or load
- No sound, no magic stone



```
You guessed 0 x 1.
You found a Mugwump.
Nearest is 8 squares away to the SouthEast.
```

```
Tries: 16
Found: 2
```

```
  |0|1|2|3|4|5|6|7|8|9
 0|o| |o| | | |o| | |
 1|x| | | | | | | | |
 2|o| | |o| | | |o| |
 3|o| | |x|o|o| |o| |
 4| | | | | | |o|o|o|
 5| |o| | | | | | | |
 6| | | | | | | | | |
 7| | | | | | | | | |
 8| | | | | | | | | |
```

Illustration 21: Screenshot - Flash version

Different gameplay

- Bushes are text
- No player sprite

Bushes were drawn as text for this version because there was an issue with importing graphics and creating instances of them using scripting (Illustration 21). This is solvable given time and effort. Neither menues nor save and loading of game data was explored.

The version of flash contains a bug so that locations of visible bushes and hidden Mugwumps do not match. This can be fixed, but the trial expired before the solution could be found so the current version it is only a working prototype.

4.3 Key Features

The game is simple. There are a number of visible objects (bushes) placed in random locations on a map. Mugwumps (hidden objects) are then placed inside a bush. The player guesses the location of the Mugwumps by using the mouse to click on a bush. Random number generators from the game engines were used to choose where the objects landed. Each game engine handles the basic game loop (updating graphics, updating objects and such) in a different way. Objects (or items) were handled slightly differently between engines. The rest of the game logic is nearly the same between engines.

4.3.1 Game Loop

In XNA there are five basic methods for handling the game loop. *Initialize()* is run at the start, checking for any requirements and loading non-graphic components. *LoadGraphics()* and *UnloadGraphics()* handle loading and unloading graphic components. *Update()* handles the game logic, updating the game as needed, and finally *Draw()* draws the frame. *Update()* is called repeatedly in a loop untill the game ends. Event handlers, such as mouse clicks, are separately handled by *HandleInput()*. For the Mugwump game *Update()* calls for changes in the game status based on variables, such as updating the status of the player sprite depending on it's location. *HandleInput()* waits for a mouse click or a keyboard press to respond to.

In Panda3D the game logic is run once and then stops with the screen open and refreshes the graphics on a frame rate. Unless the player sets up a loop or adds an event handler, the screen does not appear to change at all. The game screen is redrawn at a set frame rate but does not update the graphics unless specifically told to. For the Mugwumps game an event handler for the mouse waits for it to be clicked to push the game forward and an event handler for the keyboard waits for ESC to exit the game. Flash is similar to Panda3D. The game is pushed forward by events. It is possible to have timed events as well as mouse and keyboard events.

For Mugwumps in Panda3D and Flash, the main event handler for pushing the game forward is 'on mouse click'. When a user presses the mouse button it is checked where it happened and what to do based on that, similar to XNA, and then the graphics and Muggy hints are updated as needed.

4.3.2 Visible and Hidden Objects

In XNA the objects, or bushes, have a specific class named Items. It keeps track of the status of the object (if it has a hidden object or not, if it has been searched or not) using an integer. Another way would have been to use booleans for that. The class also handles graphics and keeps track of the location of the object on the map. The game chapter creates an array of items equal in size to how many are visible and places them randomly on the map using a random number generator. Then hidden objects, Mugwumps, are placed within n-number of visible objects using a random number generator and the status of those objects changed to containing hidden.

In Panda this is similar, a specific class for items keeps track of their location and graphic and the game creates an array of items. Flash on the other hand is more similar to the Java version of the game. It uses only an array for the map and marks the status of the map square, wether it is occupied and with what. It would be possible to create a separate class in flash to handle items given more time and knowledge of action script.

4.4 Coding Convensions

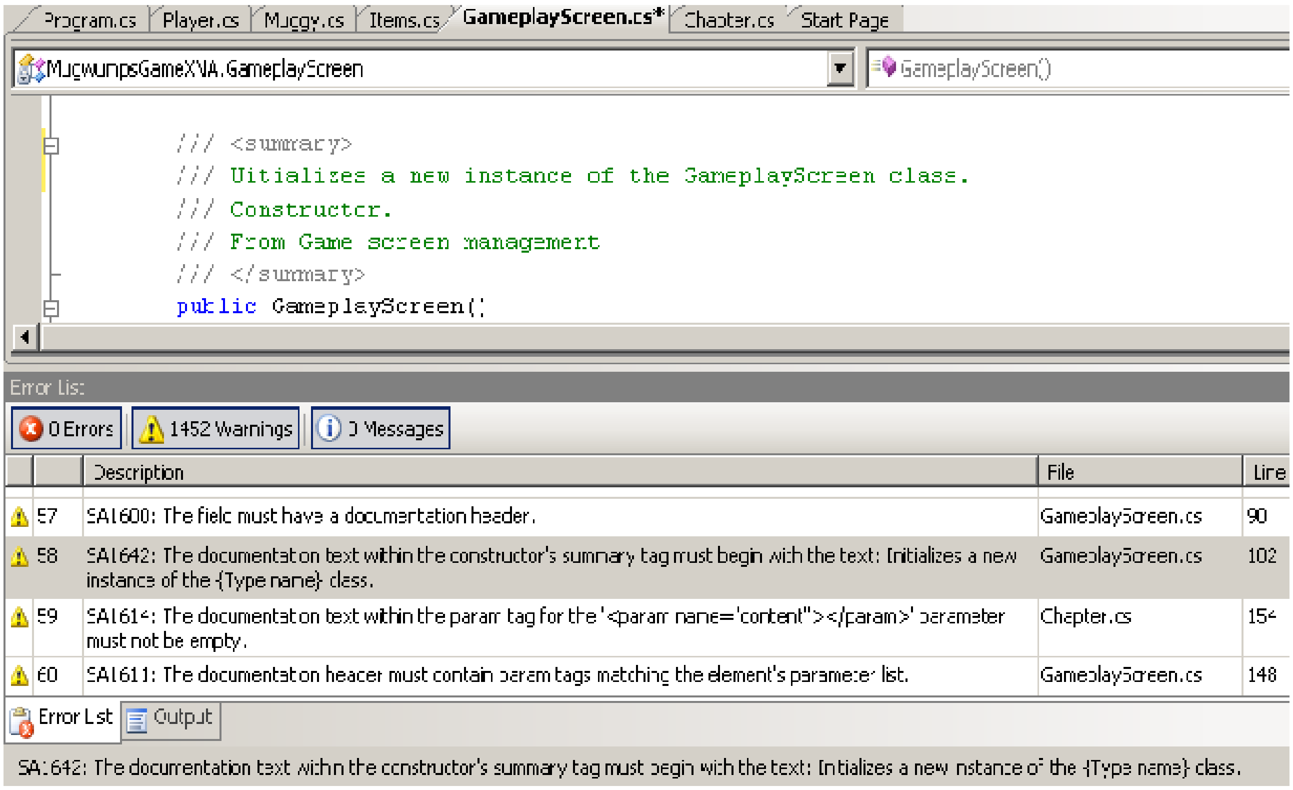
All code was written with the same basic ideas at first then fixed to the coding conventions of each language. Normally the coding conventions for a specific language should be used from the start of programming but this reverse order was done to see the difference between them clearer.

The basic rules followed.

- Methods begin with upper case first letter, variables lower case first letter.
- Names are descriptive, e.g. muggySays for a string containing what Muggy says.
- Comments for each method and group of variables at the minimum. Comment if – else statements.

4.4.1 XNA

For XNA a specific addon was installed to Microsoft Visual C# Express. The addon is called styleCop and is a free Microsoft product. StyleCop prints out warnings for any break from coding conventions. It ranges from recommending changing from Hungarian notations (e.g. iNumber for an integer variable) to keeping track of if documentation is adequit (e.g. Specific wording of constructor summary).



Program.cs | Player.cs | Muggy.cs | Items.cs | GameplayScreen.cs* | Chapter.cs | Start Page

MugwumpsGameXNA.GameplayScreen | GameplayScreen()

```
        /// <summary>
        /// Uitializes a new instance of the GameplayScreen class.
        /// Constructor.
        /// From Game screen management
        /// </summary>
        public GameplayScreen()
```

Error List

0 Errors | 1452 Warnings | 0 Messages

		Description	File	Line
⚠	57	SA1600: The field must have a documentation header.	GameplayScreen.cs	90
⚠	58	SA1642: The documentation text within the constructor's summary tag must begin with the text: Initializes a new instance of the {Type name} class.	GameplayScreen.cs	102
⚠	59	SA1614: The documentation text within the param tag for the '<param name='content"></param>' parameter must not be empty.	Chapter.cs	154
⚠	60	SA1611: The documentation header must contain param tags matching the element's parameter list.	GameplayScreen.cs	148

Error List | Output

SA1642: The documentation text within the constructor's summary tag must begin with the text: Initializes a new instance of the {Type name} class.

Illustration 22: XNA: styleCop - First Run

Code was fixed according to StyleCop.

- Hungarian notations removed, pPlayer renamed to player for example.
- /// <summary> description </summary> added above all methods and classes instead of // description
- Spaces and linebreaks fixed. e.g. Linebreak before a single line comment, linebreak after closing an else statement.
- Order of methods and variables. Public before private.

Before fixing the code StyleCop reported 1452 warnings. After fixing Style Cop reports 810 warnings, only 120 of those directly from the Mugwumps game.

Warnings that were ignored were

- Warnings related to the XNA package (spaces and linebreaks) and Screen Manager.
- Warnings related to documentation requirement (too few words). This is mostly done due to lack of time, though it would be prefered to complete the documentation properly.

4.4.2 Panda3D

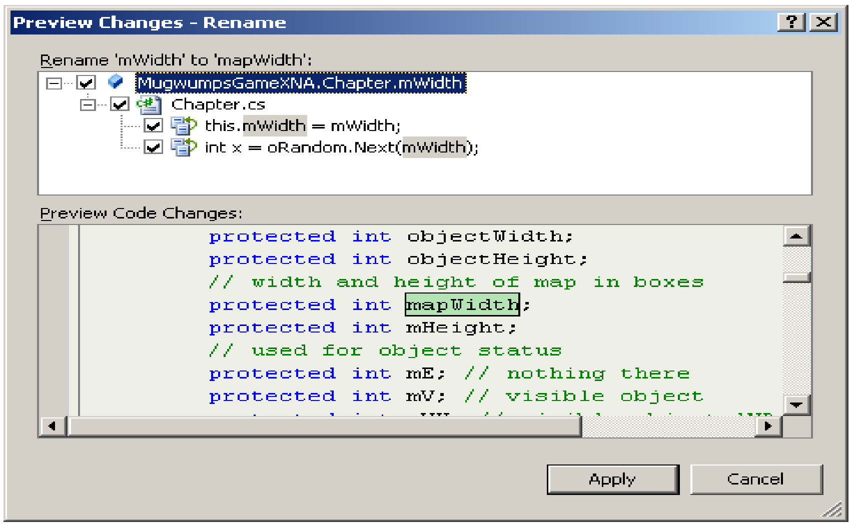
For Panda3D, PEP 8 – Style Guide for Python Code (python.org 2010) was used. It is a short and simple list of rules. The python code for Mugwumps is short as well so it was corrected manualy according to the guide. It took one third of the time it took to correct the C# code with StyleCop because the changes needed to be done were fewer and the original code closer to the style guide.

4.4.3 Flash

For Flash a whitepaper from Adobe on action script was used (Adobe 2010). It contains the do's and don't's on action script standards. Code was corrected manually since the list is simple and code short.

4.5 Static Analysis and Refactoring

All three engines have static analysis and refactoring tools available, though not all IDE's chosen have them to start with. Both XNA and Panda3D IDEs had refactor and rename built in.


Preview Changes - Rename

Rename 'mWidth' to 'mapWidth':

```
MugwumpsGameXNA.Chapter.mWidth
  Chapter.cs
    this.mWidth = mWidth;
    int x = oRandom.Next(mWidth);
```

Preview Code Changes:

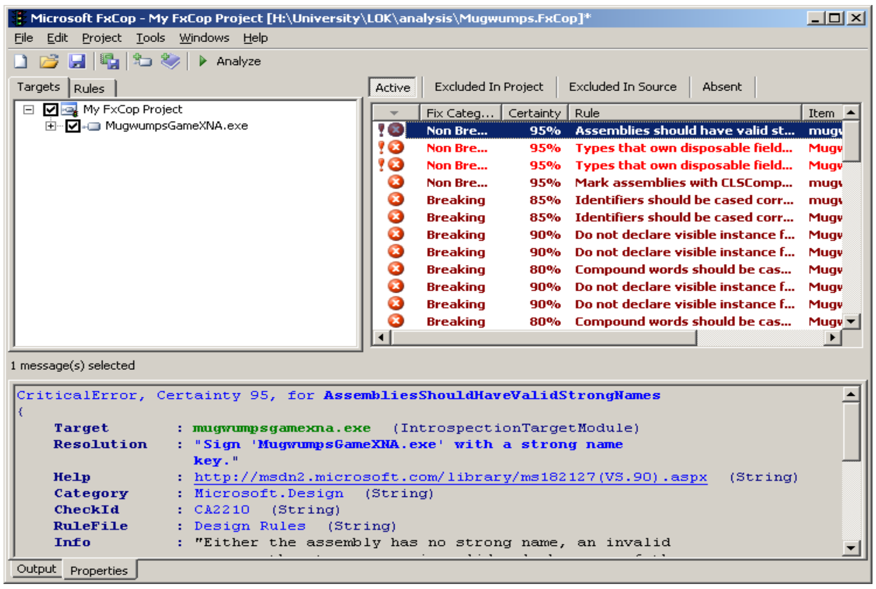
```
protected int objectWidth;
protected int objectHeight;
// width and height of map in boxes
protected int mapWidth;
protected int mHeight;
// used for object status
protected int mE; // nothing there
protected int mV; // visible object
```

Illustration 23: XNA - Visual C# 2008 Express Refactor:Rename

4.5.1 XNA

For C# a basic rename and extract method is in Visual Express (Illustration 23). The professional edition of this IDE has more tools for static analysis. For this project, another free Microsoft product was used named FxCop. FxCop checks for protection of methods and variables and other possible runtime issues.

Running FixCop after using StyleCop gives 58 warnings. Several errors are related to XNA and Screen Manager so ignored. Errors from the Mugwumps are unused methods, the getters and setters specifically. Since the game is supposed to be expanded these will be left (Illustration 24).


Microsoft FxCop - My FxCop Project [H:\University\LOK\analysis\Mugwumps.FxCop]*
File Edit Project Tools Windows Help
Analyze
Targets Rules
My FxCop Project
MugwumpsGameXNA.exe
Active Excluded In Project Excluded In Source Absent
Output Properties

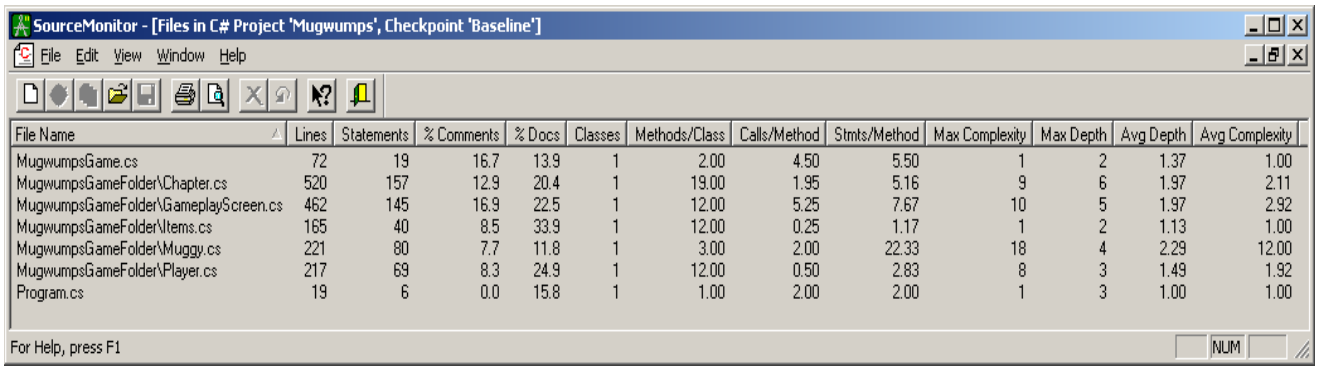
Fix Categ...	Certainty	Rule	Item
Non Bre...	95%	Assemblies should have valid st...	mugv
Non Bre...	95%	Types that own disposable field...	Mugv
Non Bre...	95%	Types that own disposable field...	Mugv
Non Bre...	95%	Mark assemblies with CLSComp...	mugv
Breaking	85%	Identifiers should be cased corr...	mugv
Breaking	85%	Identifiers should be cased corr...	Mugv
Breaking	90%	Do not declare visible instance f...	Mugv
Breaking	90%	Do not declare visible instance f...	Mugv
Breaking	80%	Compound words should be cas...	Mugv
Breaking	90%	Do not declare visible instance f...	Mugv
Breaking	90%	Do not declare visible instance f...	Mugv
Breaking	80%	Compound words should be cas...	Mugv

1 message(s) selected

```
CriticalError, Certainty 95, for AssembliesShouldHaveValidStrongNames
{
    Target       : mugwumpsgamexna.exe  (IntrospectionTargetModule)
    Resolution   : "Sign 'MugwumpsGameXNA.exe' with a strong name
                   key."
    Help         : http://msdn2.microsoft.com/library/ms182127(VS.90).aspx  (String)
    Category     : Microsoft.Design  (String)
    CheckId      : CA2210  (String)
    RuleFile     : Design Rules  (String)
    Info         : "Either the assembly has no strong name, an invalid
```

Illustration 24: XNA: FxCop - Warnings

SourceMonitor is a free product from Campwood Software. It counts, among other things, lines of codes and comment density in C# (Illustration 25). It can be used for other languages as well, though unfortunately neither Python nor Action Script.


SourceMonitor - [Files in C# Project 'Mugwumps', Checkpoint 'Baseline']
File Edit View Window Help
For Help, press F1
NUM

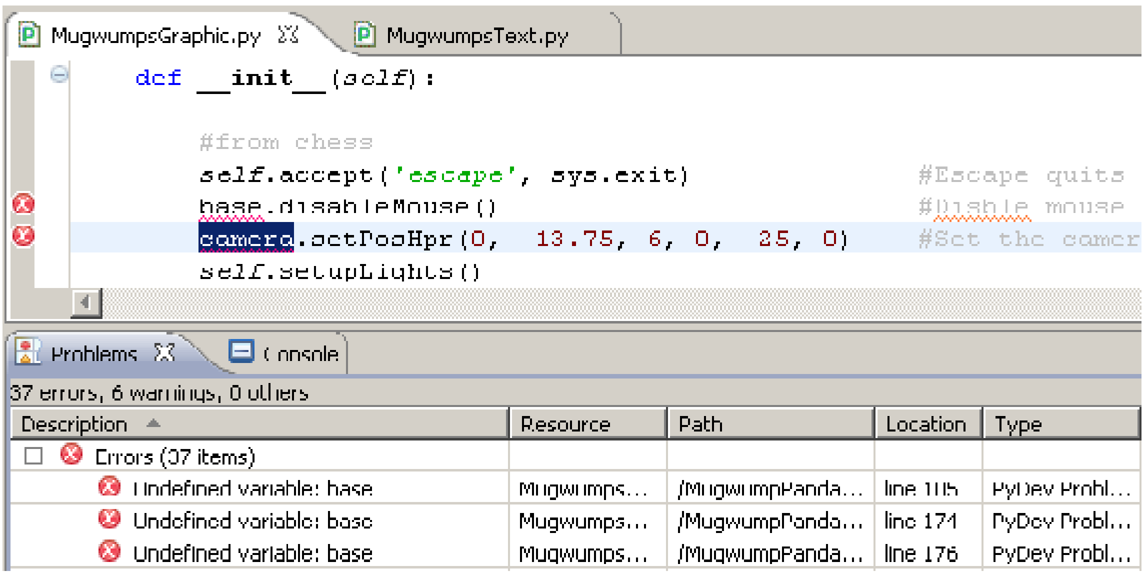
File Name	Lines	Statements	% Comments	% Docs	Classes	Methods/Class	Calls/Method	Stmts/Method	Max Complexity	Max Depth	Avg Depth	Avg Complexity
MugwumpsGame.cs	72	19	16.7	13.9	1	2.00	4.50	5.50	1	2	1.37	1.00
MugwumpsGameFolder\Chapter.cs	520	157	12.9	20.4	1	19.00	1.95	5.16	9	6	1.97	2.11
MugwumpsGameFolder\GameplayScreen.cs	462	145	16.9	22.5	1	12.00	5.25	7.67	10	5	1.97	2.92
MugwumpsGameFolder\Items.cs	165	40	8.5	33.9	1	12.00	0.25	1.17	1	2	1.13	1.00
MugwumpsGameFolder\Muggy.cs	221	80	7.7	11.8	1	3.00	2.00	22.33	18	4	2.29	12.00
MugwumpsGameFolder\Player.cs	217	69	8.3	24.9	1	12.00	0.50	2.83	8	3	1.49	1.92
Program.cs	19	6	0.0	15.8	1	1.00	2.00	2.00	1	3	1.00	1.00

Illustration 25: XNA: SourceMonitor – Final Run

Comment and documentation density in Mugwumps XNA version is from 19.5% up to 42.2% which would be acceptable. Most of the classes are between 1 and 3 in complexity but Muggy.cs which handles what Muggy says is 12.00. It would be advisable to find a better way to handle what Muggy says, if possible.

4.5.2 Panda3D

PyDev has syntax analysis, refactoring, debugging and code completion built in. There is a known problem with the code completion and analysis with Panda3D, it does not seem to recognize all Panda3D library components. Final version of the Mugwump code includes 37 errors and 6 warning that are false, they relate to Panda3D libraries (Illustration 26).



```
    def __init__(self):

        #from chess
        self.accept('escape', sys.exit)               #Escape quits
        base.disableMouse()                           #Disable mouse
        camera.setPosHpr(0, 13.75, 6, 0, 25, 0)       #Set the camer
        self.setupLights()
```

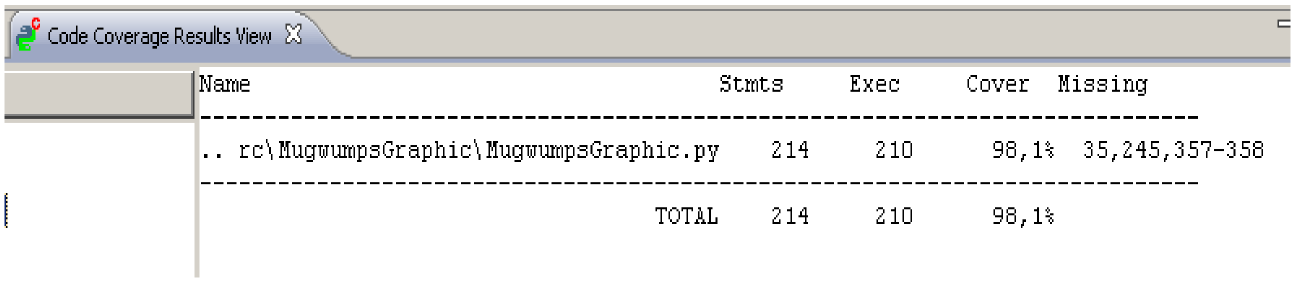
37 errors, 6 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (37 items)				
Undefined variable: base	Mugwumps...	/MugwumpPanda...	line 115	PyDev Probl...
Undefined variable: base	Mugwumps...	/MugwumpPanda...	line 174	PyDev Probl...
Undefined variable: base	Mugwumps...	/MugwumpPanda...	line 176	PyDev Probl...

Illustration 26: Panda3D: PyDev - False Errors and Warnings

A good feature of PyDev is the To-do list. If a comment begins with TODO it will be moved to a specific list that the programmer can use to keep track of what is left.

Code Coverage is a feature of PyDev wich is extremely helpfull in testing code. It measures how much of the code was covered in a run test and notes the lines of code that were not covered. See Illustration 27 for the result of a random click run.



```
Name                                        Stmts     Exec     Cover   Missing
-----------------------------------------------------------------------------
.. rc\MugwumpsGraphic\MugwumpsGraphic.py      214      210      98,1%   35,245,357-358
-----------------------------------------------------------------------------
                                   TOTAL      214      210      98,1%
```

Illustration 27: Panda3D: PyDev - Code Coverage, Random Click

4.5.3 Flash

No static analysis was done on the Flash version because the trial had run out. There are available for purchase various programs and tools to perform analysis with.

5 Evaluation

5.1 HCI Evaluation

Informal think-aloud sessions were performed at the beginning and at the end of the implementation phase for the XNA version of Mugwumps. This was to test the user interface and game experience and adjust. One session was done for each of the Panda3D and Flash versions once they were nearly complete. The reason for the extra session for XNA at the beginning of the phase was to get rid of the most obvious faults in the design before much coding was done. The think aloud sessions were done with four different people, Fjóla María (FM), Ólöf Hörn (ÓH), Hlynur (H) and Örvar (Ö). FM and ÓH were together in the session at the start of implementation for XNA and end of implementation for each version. H and Ö were together for a session after implementation in XNA.

5.1.1 Beginning of Implementation

At the beginning of implentation only a rough version of the game was used. A grid with coloured boxes representing bushes were visible as well as Muggy's hints.

It was clear from the session that an avatar (sprite) was needed for the player to feel connected to what was happening on the screen and to see where the last mouse click was. Muggy's hints were misleading, 'Listen to..' was used, prompting the player to ask if the sound was on. Muggy also said 'Go search..' rather than specificly guide the player to 'Search the bushes.' leading to confusion on how to play the game. These issues were noted down and fixed in the XNA version and kept in mind for the Panda3D and Flash.

5.1.2 XNA

XNA session at the end of implementation was more succesfull. The game was found to be simplistic but easy to follow if the hints were read. It was unclear what to do if the player did not read the hints, but since that is part of the game – to read and learn to follow hints – this is acceptable. It is possible to complete the game by clicking randomly but the game score will usually be poor.

A bug with the player sprite came up, it did not always go all the way to where the user clicked with the mouse. Attempts to fix this were unsuccsesfull.

5.1.3 Panda3D

Playing a game with pawns instead of bushes and coloured squares instead of the pawns changing colour turned out to be confusing. Lacking the player sprite and a picture of Muggy to emphatise with made the game play less fun. It would be possible to change the colour of the pawns instead of the squares but that would only be a minor improvement so was not done.

The game used the mouse location in relation to the squares of the chessboard instead of the pawns wich led to what appears as a bug: if the top of a pawn's head is clicked, the game reads it as the square above the pawn (if there is one) or nothing. It would be better to link the mouse click with what pawn was clicked rather than square.

5.1.4 Flash

No HCI evaluation was done on Flash.

5.2 Software Testing

While the game was being programmed, after each new implementation it was run and tested. The testing covered basic operations, such as when adding the event handler for a mouse click the user clicked randomly on the screen and observed feed back in text where the click was. This is not an organized way of testing and can not be relied on to find the most errors. After each version was done, the original use cases were used to test and then new were made based on the functionality of the version. Unit testing was planned but not performed.

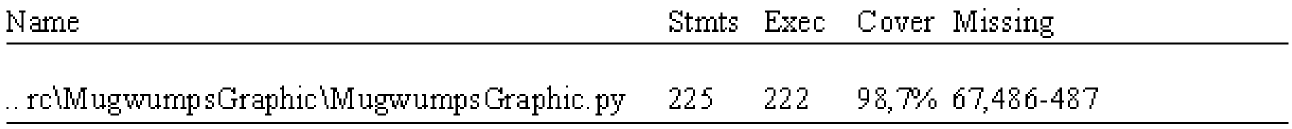
5.2.1 XNA

The XNA version was tested using use cases and new were created (included in Appendix F). The version was also tested by the programmer using different tactics to complete the game (random clicking, organized left to right clicking all squares and finally carefully following hints). There is an issue with the player sprite not moving to the mouse click at all times but that was the only visible bug. The version was not tested on other computers or with other configurations.

The testing coverage is not enough. More should be done in terms of software compatability testing as well as unit testing. Unit testing is not available in Visual C# Express edition, but it is available in the professional edition as well as there are third party software available.

5.2.2 Panda3D

Panda3D version was tested using use cases and new were created (included in Appendix F). PyDev's code coverage was enabled while doing the use case testing and showed 98,7 % test coverage.



Name	Stmts	Exec	Cover	Missing
..rc\MugwumpsGraphic\MugwumpsGraphic.py	225	222	98,7%	67,486-487

Illustration 28: Panda3D - Code Coverage

The code missing from the test are two different methods. One gives a value on the z-axis where an object should be if it followed the mouse pointer. The other changes the colour of the pawns. Both are from the chess tutorial and are kept in, incase they can be used later on.

The 98,7% code coverage gives hope that errors occurring is less likely but it does not cover different software enviroments. Unit testing is available in PyDev and would be advicable to test different inputs but was not done due to lack of time. The Panda3D version of the game was also tested using different tactics to complete the game (random clicking, organized left to right clicking all squares and finally carefully following hints) and no errors were encountered.

5.2.3 Flash

No organized testing was performed. Random testing of playing the game revealed that the visible bushes and where the Mugwumps are hidden does not match. While fixing that the trial ran out so that the flash file was not recompiled with the fixes. The action script file does contain an attempt to fix this issue but has not been complied into a flash file.

5.3 Game Engine Comparison

To compare the three engines a few factors were looked into and evaluated. The comparison is in part based on numbers and in part based on nominal attributes which will be assigned by the writer's experience. This comparison is not complete but does give an idea of the qualities and weaknesses of each engine.

5.3.1 Programing experience

XNA turned out to be easy to use, the IDE pops up with suggestions for words and error checks as the code is typed. The PyDev in Eclipse IDE is alright, but lacks the word completion and as-you-type error checking that XNA has (PyDev due to an error with the Panda3D package). Adobe flash's

IDE is big and heavy. It takes much longer to get used to it and it lacks access to refactoring and word completion that the others have. You can get addons for that though.

The code in all three languages is easily readable. The use of spaces instead of brackets, {}, in python does provide some confusion and it is easier to make a mistake by forgetting to add the fourth space rather than forgetting to add a bracket since the space is 'invisible'.

All in all both XNA and Panda3D are flexible and easy to start with engines that accomodate a programmer well but Flash IDE is more suited for a graphic point and click development.

	XNA	Panda3D	Flash
Code readability (nominal)	Good.	Good, but using spaces instead of {} can confuse.	Good.
Functionality (nominal, list)	Support for 2D and 3D Basic physics	Support for 2D and 3D Basic physics	Support for 2D and 3D
IDE qualities (nominal, list)	Simple Code completion Error checking while typing Basic refactoring	Simple Basic refactoring Code testing coverage Unit testing	
IDE weaknesses (nominal, list)	No unit testing Memory heavy	Bug in code completion and error checking	Big and heavy

Table 1: Comparison - Programing Experience

5.3.2 Support

The official help documentation is decent for all three engines. On the official webpages of each is a basic introduction to the engine, detailed information on functions and tutorials. Panda3D has an extensive library of tutorial with the instal folder, where the programmer can view the well commented code behind mini programs.

	XNA	Panda3D	Flash
Official support (nominal)	Very good	Very good	Very good
Community support (nominal)	Extremely good	Extremely good	Confusing

Table 2: Comparison - Support

The XNA has an official community page, XNA Creators Club, that is organized and it seems to be easy to find solutions or help to solve problems there. Panda3D has a large selection of solutions provided by the community on forums hosted on Panda3D's official site. Users can ask for aid on these forums as well and they seem fairly active. The user generated help for Flash – various tutorials and how to's – are so many and unfocused that it takes a while to find something useful.

There is no one community site for Flash like the other two, instead a developer has to rely on skills in searching the internet for various forums and tutorial hubs. Much of the flash tutorials are available for payment only.

The easily available help for XNA and Flash shows that it makes a difference wether the software developer provides grounds for the software users to communicate or not.

5.3.3 Time consumed

The implementation time is the time it took to create a basic version of the game and adding graphics. Before the implementation, time was used to familiarize with the engines and code by going through a 'hello world' tutorial provided by the official sites and then creating a text version of the Mugwumps game. Creating the text version could be counted in as implementation time since most of the game logic was written there, but it can also fall under familiarizing with a new scripting language. The text version was almost a dirrect recreation of a Java version of Mugwumps.

The actual time to program a basic graphic game is similar in XNA and Panda3D but trying to make the graphics 2D in Panda3D swallowed up a considerable time. In the end a 3D chess tutorial was used for graphics in Panda3D, but making a 2D game is not impossible. Figuring out how to create labels and such in Flash took a while, but the basic coding took noticably shorter. This can be attributed to the similarity between C# and Action Script 3.0, making the writing of code for Flash nearly a dirrect write-up of the XNA version.

Time in hours	XNA	Panda3D	Flash
Tutorials	2,5	8,5	7,5
Simple Text version	7	5,5	2,5
Basic version	10	8,5	2,5
Graphics	3,5	10	7
Refactoring	13	5	1

Table 3: Comparison - Time consumed

Developing a game in Panda3D would likely take less time than in XNA, provided that the use of 3D graphics takes the same amount in both. For a 2D game XNA is a better choice because it is easy to work with but for a full 3D game Panda3D's shorter code and eager community help should be considered.

5.3.4 Runtime resources

File size for the different programs varies. It should be noted that the XNA version does have a little more functionality and that such a small program does not give an idea of how the file size scales with a larger program.

	XNA	Panda3D	Flash
File size (numerical)	8 MB	1.9 MB	138 KB
Run time memory consumption (numerical)	27 MB	61 MB	
Load time (numerical)	9 sec.	2 sec.	
Run time errors and glitches (nominal, list)	Player sprite does not go all the way to mouse click at times.	Not able to click pawns to guess location.	Bushes and hidden Mugwumps do not match in location.

Table 4: Comparison - Runtime resources

5.3.5 Code Metrics

Finding and using a code metrics measurement for Panda3D and Flash proved unsuccesfull. It is obvious from reading the code that comment density in XNA is greater than in the other two and that the requirements for documentation is greater from the official style guides. This and having more features could explain some of the difference between Panda3D and XNA. Flash is noticably shorter but the code there does not deal with the graphics, unlike the other two.

To test the difference between the Panda3D and XNA versions better, the complete Panda3D version and the classes GameplayScreen and Chapter from the XNA version were stripped of all comments and measured. The reason for choosing these two classes only is because they contain most of the basic functionality and least of the additional functions. The XNA code proved still to be much longer than the Panda3D one, which is in line with previous comparisons of programming languages that showed that C code is longer than Python. Those researches showed that a longer code takes longer to create and because of this a Panda3D game should take shorter than an XNA game of the same type to create.

	XNA	Panda3D	Flash
Lines of Code (numerical)	1676*	492	287
Lines of Code – stripped down	570	306	

* XNA Lines of Code does not include the screen manager.

Table 5: Comparison - Code metrics

6 Conclusion

The purpose of this project was to create a game in three different game engines and compare the programing process and results. The reason why this is interesting is because the choices of a starting game creator are overwhelming and difficult to know what tools are best to start with. A comparison might help with this choice. This project tackled the choice from the point of view of someone who wants to practice programming skills as well as development skills. It looked at the pro's and con's of many different languages and platforms and selects three to build a game in. Then an old BASIC game called Mugwumps was created in each engine.

During this I learned the basics in programming in C#, Python and Action Script 3.0. I learned to use the debugging mechanisms of PyDev and refactoring of Visual C# Express edition as well as StyleCop to correct C# coding style. I got a taste of how 3D graphics are programmed in a game, though I did not succesfully manipulate them. I became familiar with the three game engines; XNA, Panda3D and Adobe Flash. I also got practice in managing my time, making schedules and try to stick with them.

The time table was not followed well enough. The work became two weeks behind after being sick so that the time to create Panda3D and Flash was reduced. Most of the lost time on Panda3D was worked up but Flash was not worked on hard enough so that the trial expired before it was completed. I learned that for a project like this it is not wise to use a trial of a software. Even if the version had been completed on time, it would have been difficult to return to the code or tools to check on other things or display the work afterwards.

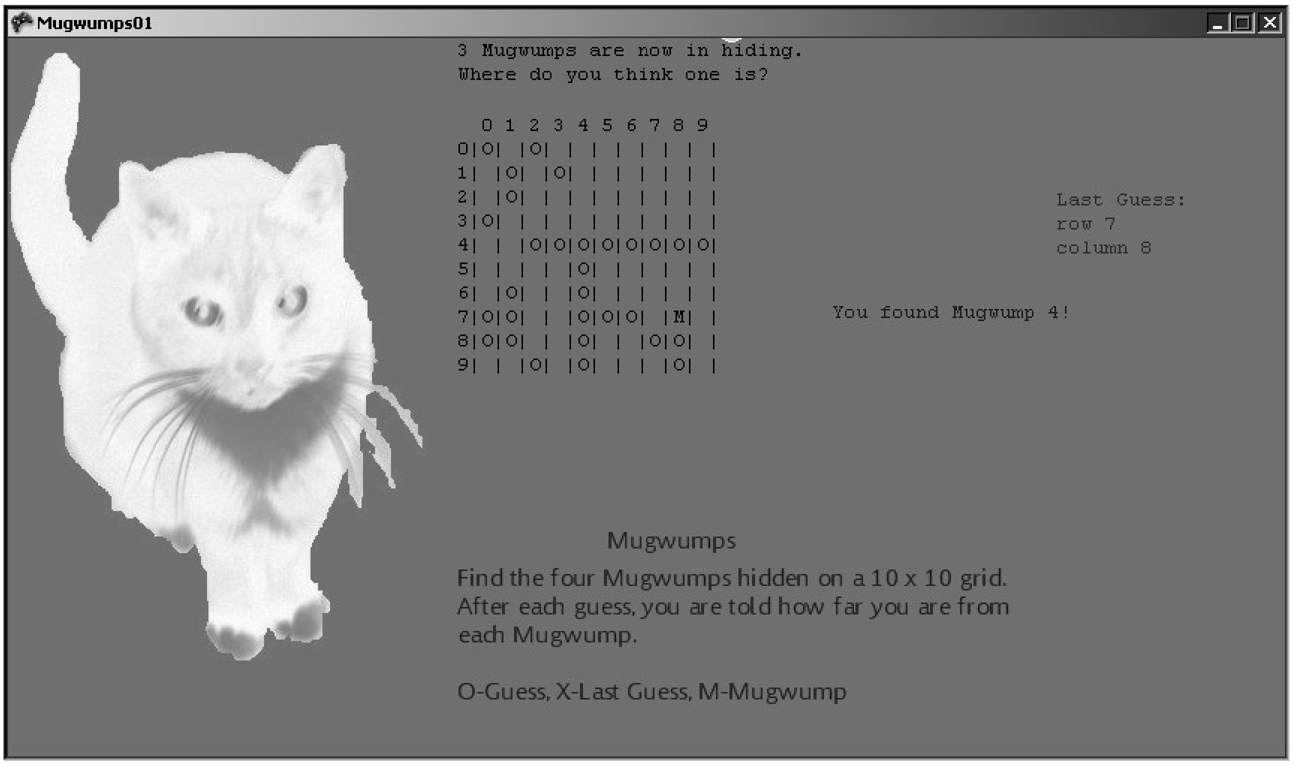
Having three engines to compare gives some dynamic to the comparison but it is very difficult to do within the time frame given. It would have been more efficient to focus on only two, such as Panda3D and XNA because they are similar, giving more time to programing and analysing to each. Flash also turned out to be a difficult engine to compare to the other two because of it's IDE graphical nature.

Future work with this project could lead to two things. One would be to complete the game and publish it. If it is to be a 2D game it would be easiest to complete it in XNA since that is furthest along. If it is to be 3D, such as paperdolls in a 3D field, then Panda3D would be interesting to look at because of the open source nature of the engine, wide support and (in theory) shorter programing time in Python. Flash is less interesting though it would be possible to find a free flash editor to create the game in.

The other possibility of future work is to do more research into the difference of game engines and their languages. This could be done with different game engines, with more programmers tackling the same program and/or taking a closer look at the difference of nature between XNA and Panda3D. It would be interesting to compare the C++ version of Panda3D with the C++ version of XNA to see the difference between engines only, rather than both engine and language effecting the difference.

This was a very interesting project to take on. More time could have been devoted to it and decisions at the start to take three rather than two engines could have been different but it does provide a basic comparison between three different engines and a demo of a playable video game.

Appendix E - Prototype



```
3 Mugwumps are now in hiding.
Where do you think one is?

  0 1 2 3 4 5 6 7 8 9
0|O| |O| | | | | | | |
1| |O| |O| | | | | | |
2| |O| | | | | | | | |
3|O| | | | | | | | | |
4| | |O|O|O|O|O|O|O|O|
5| | | | |O| | | | | |
6| |O| | |O| | | | | |
7|O|O| | |O|O|O| |M| |
8|O|O| | |O| | |O|O| |
9| | |O| |O| | | |O| |

Last Guess:
row 7
column 8

You found Mugwump 4!
```

Mugwumps

Find the four Mugwumps hidden on a 10 x 10 grid.
After each guess, you are told how far you are from each Mugwump.

O-Guess, X-Last Guess, M-Mugwump

```
#region File Description
//-----------------------------------------------------------------------------
// GameplayScreen.cs
//
// Microsoft XNA Community Game Platform
// Copyright (C) Microsoft Corporation. All rights reserved.
//-----------------------------------------------------------------------------
#endregion

#region Using Statements
using System;
using System.Threading;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
#endregion

namespace GameStateManagement
{
    /// <summary>
    /// This screen implements the actual game logic. It is just a
    /// placeholder to get the idea across: you'll probably want to
    /// put some more interesting gameplay in here!
    /// </summary>
    class MugwumpsStart : GameScreen
    {
        #region Fields
```

```
ContentManager content;
SpriteFont gameFont;
SpriteFont courier;
Color backColor = Color.CornflowerBlue;

Vector2 playerPosition = new Vector2(100, 100);
//Vector2 enemyPosition = new Vector2(100, 100);

//Random random = new Random();

#endregion

#region Initialization

// initialize for MugWump
Random jRandom = new Random();
        int[,] p = new int[4,2];
        int[,] map = new int[10,10];
        int remaining = 4;
int tries = 0;
        int m, n;
        Boolean found;

/// <summary>
/// Constructor.
/// </summary>
public MugwumpsStart()
{
  TransitionOnTime = TimeSpan.FromSeconds(1.5);
  TransitionOffTime = TimeSpan.FromSeconds(0.5);
}

Texture2D odur;
Texture2D dot;
Vector2 spritePosition = Vector2.Zero;
Vector2 dotPosition = Vector2.Zero;
SpriteFont font;
String mapString = "";
String guessString = "";

/// <summary>
/// Load graphics content for the game.
/// </summary>
public override void LoadContent()
{

  if (content == null)
    content = new ContentManager(ScreenManager.Game.Services, "Content");

  gameFont = content.Load<SpriteFont>("gamefont");
  courier = content.Load<SpriteFont>("courier");

  odur = content.Load<Texture2D>("odur");
  font = content.Load<SpriteFont>("SpriteFont1");
  dot = content.Load<Texture2D>("dot");

  // A real game would probably have more content than this sample, so
  // it would take longer to load. We simulate that by delaying for a
```

```
        // while, giving you a chance to admire the beautiful loading screen.
        //Thread.Sleep(1000);

        //  hide mugwumps
        for (int i = 0; i < 10; i++)
        {
            for (int j = 0; j < 10; j++)
            {
                map[i,j] = 0;
            }
        }
        for (int i = 0; i < 4; i++)
        {
            do
            {
                for (int j = 0; j < 2; j++)
                {
                    p[i,j] = jRandom.Next(10);
                }
            }
            while (map[p[i,0],p[i,1]] != 0);
            map[p[i,0],p[i,1]] = 1;
        }
        for (int i = 0; i < 4; i++)
        {
            map[p[i,0],p[i,1]] = 0;
        }


        // once the load has finished, we use ResetElapsedTime to tell the game's
        // timing mechanism that we have just finished a very long frame, and that
        // it should not try to catch up.
        ScreenManager.Game.ResetElapsedTime();
    }


    /// <summary>
    /// Unload graphics content used by the game.
    /// </summary>
    public override void UnloadContent()
    {
        content.Unload();
    }


    #endregion

    #region Update and Draw


    /// <summary>
    /// Updates the state of the game. This method checks the GameScreen.IsActive
    /// property, so the game will stop updating when the pause menu is active,
    /// or if you tab away to a different application.
    /// </summary>
    ///

    Boolean endIt = false;
```

```
public override void Update(GameTime gameTime, bool otherScreenHasFocus,
                                           bool coveredByOtherScreen)
{
   base.Update(gameTime, otherScreenHasFocus, coveredByOtherScreen);

   if (IsActive)
   {
      PrintMap();
      UpdateMouse();

      if (remaining == 0)
      {
         EndHunt();
         endIt = true;
      }

      if (endIt)
      {
         ScreenManager.AddScreen(new EndMenuScreen(tries), ControllingPlayer);
      }
   }
}

String endString = "";

protected void EndHunt()
{
   endString = "You found all the Mugwumps with " + tries + " tries! \nGood job!";
}

protected void PrintMap()
{
   // draw current game board grid
   mapString = "  0 1 2 3 4 5 6 7 8 9\n";
                                            for (int i = 0; i < 10; i++)
                                            {
                                                        mapString = mapString + (i) +"|";
                                                        for (int j = 0; j < 10; j++)
                                                        {
                                                                    switch (map[i,j])
                                                                    {
                                                                                case 0:
                                                                                mapString = mapString + " |";
                                                                                break;
                                                                                case 1:
                                                                                mapString = mapString + "O|";
                                                                                break;
                                                                                case 2:
                                                                                mapString = mapString + "X|";
                                                                                //map[i,j] = 1; get rid of x elsewhere
                                                                                break;
                                                                                case 3:
                                                                                mapString = mapString + "M|";
                                                                                break;
                                                                    }
                                                        }
                                            mapString = mapString + "\n";
                                            }
```

```
    }

    Boolean pressMouse = false;

    protected void UpdateMouse()
    {
       MouseState current_mouse = Mouse.GetState();

       // The mouse x and y positions are returned relative to the
       // upper-left corner of the game window.
       int mouseX = current_mouse.X;
       int mouseY = current_mouse.Y;

       dotPosition.X = current_mouse.X;
       dotPosition.Y = current_mouse.Y;

       if (current_mouse.LeftButton == ButtonState.Released)
          pressMouse = false;
       // find mugwumps!
       if ((current_mouse.LeftButton == ButtonState.Pressed )&& (pressMouse == false))
       {
          pressMouse = true;
          // The mouse x and y positions are returned relative to the
          // upper-left corner of the game window.
          int nX = current_mouse.X;
          int mY = current_mouse.Y;

          guessMugwump(nX, mY);
       }

       // Change background color based on mouse position.
       //backColor = new Color((byte)(mouseX / 3), (byte)(mouseY / 2), 0);
    }

    /// <summary>
    /// Lets the game respond to player input. Unlike the Update method,
    /// this will only be called when the gameplay screen is active.
    /// </summary>
    public override void HandleInput(InputState input)
    {
       if (input == null)
          throw new ArgumentNullException("input");

       // Look up inputs for the active player profile.
       int playerIndex = (int)ControllingPlayer.Value;

       KeyboardState keyboardState = input.CurrentKeyboardStates[playerIndex];
       GamePadState gamePadState = input.CurrentGamePadStates[playerIndex];

       // The game pauses either if the user presses the pause button, or if
       // they unplug the active gamepad. This requires us to keep track of
       // whether a gamepad was ever plugged in, because we don't want to pause
       // on PC if they are playing with a keyboard and have no gamepad at all!
       bool gamePadDisconnected = !gamePadState.IsConnected &&
                                  input.GamePadWasConnected[playerIndex];

       if (input.IsPauseGame(ControllingPlayer) || gamePadDisconnected)
       {
          ScreenManager.AddScreen(new PauseMenuScreen(), ControllingPlayer);
       }
       else
       {
          // Otherwise move the player position.
          Vector2 movement = Vector2.Zero;

          if (keyboardState.IsKeyDown(Keys.Left))
              movement.X--;

          if (keyboardState.IsKeyDown(Keys.Right))
              movement.X++;

          if (keyboardState.IsKeyDown(Keys.Up))
              movement.Y--;

          if (keyboardState.IsKeyDown(Keys.Down))
              movement.Y++;

          Vector2 thumbstick = gamePadState.ThumbSticks.Left;

          movement.X += thumbstick.X;
          movement.Y -= thumbstick.Y;

          if (movement.Length() > 1)
              movement.Normalize();

          playerPosition += movement * 2;
       }
    }
    int oldM = 0;
    int oldN = 0;
    protected void guessMugwump(int nX, int mY)
    {
       n = (nX - 311) / 16;
       m = (mY - 68) / 16;

       guessString = "";

       if (m < 0 || m > 9 || n < 0 || n > 9)
       {
          guessString = "You must choose a place in the grid!";
       }
       else
       {
          // get rid of x
          if (map[oldM, oldN] == 2)
              map[oldM, oldN] = 1;
          oldM = m;
          oldN = n;
          switch (map[m, n])
          {
              case 1:
                  guessString = "You already looked there!";
                  break;
              case 2:
                  guessString = "You already looked there!";
                  break;
              case 3:
                  guessString = "You already found this Mugwump!";
                  break;
              case 0:
                  tries++;
                  found = false;
                  for (int i = 0; i < 4; i++)
                  {
                    if (p[i, 0] != -1)
                    {
                      if (p[i, 0] != m || p[i, 1] != n)
                      {
                        map[m, n] = 2;
                        double d = Math.Sqrt((p[i, 0] - m) * (p[i, 0] - m) + (p[i, 1] - n) * (p[i, 1] - n));
                        guessString = guessString + "You are " + d.ToString("0.00") +
                          " units from Mugwump " + (i + 1)+"\n";
                      }
                      else
                      {
                        found = true;
                        p[i, 0] = -1;
                        guessString = guessString + "You found Mugwump " + (i + 1) + "!\n";
                        remaining--;
                      }
                    }
                    if (found == true)
                    {
                      map[m, n] = 3;
                    }
                  }
                  break;
          }
       }
    }

    /// <summary>
    /// Draws the gameplay screen.
    /// </summary>
    public override void Draw(GameTime gameTime)
    {
       // This game has a blue background. Why? Because!
       ScreenManager.GraphicsDevice.Clear(ClearOptions.Target,
                                          backColor, 0, 0);

       // Our player and enemy are both actually just text strings.
       SpriteBatch spriteBatch = ScreenManager.SpriteBatch;

       //spriteBatch.Begin();

       //spriteBatch.DrawString(gameFont, "// TODO", playerPosition, Color.Green);

       //spriteBatch.DrawString(gameFont, "Insert Gameplay Here",
          //                     enemyPosition, Color.DarkRed);


       spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
       spriteBatch.DrawString(font, "Mugwumps", new Vector2(400, 325), Color.Bisque);
       spriteBatch.DrawString(font, "Find the four Mugwumps hidden on a 10 x 10 grid.\n" +
            "After each guess, you are told how far you are from \neach Mugwump.\n" +
            "\nO-Guess, X-Last Guess, M-Mugwump", new Vector2(300, 350), Color.Bisque);
        // draw cat
        spriteBatch.Draw(odur, spritePosition, backColor);
        spriteBatch.Draw(dot, dotPosition, backColor);

        spriteBatch.DrawString(courier,remaining +
              " Mugwumps are now in hiding. \nWhere do you think one is?", new Vector2(300,0), Color.White);
        // print Map
        spriteBatch.DrawString(courier, mapString, new Vector2(300, 50), Color.White);

        // press X - guess mugwump
        spriteBatch.DrawString(courier, "Last Guess:\nrow " + m + "\ncolumn " + n +
        "\ntries "+tries, new Vector2(700, 100), Color.Pink);

        spriteBatch.DrawString(courier, guessString, new Vector2(550, 175), Color.Yellow);

        spriteBatch.DrawString(courier, endString, new Vector2(500, 200), Color.Red);

        spriteBatch.End();

        // If the game is transitioning on or off, fade it out to black.
        if (TransitionPosition > 0)
            ScreenManager.FadeBackBufferToBlack(255 - TransitionAlpha);
    }


    #endregion
  }
}
```

```
'''
Created on 27.2.2010

@author: Rosa Dogg
'''

import random
import math

class Game:
  #init map
    # 0 - none
    # 1 - searched, no mugwump
    # 2 - mugwump in hiding
    # 3 - searched, mugwump found
  map = [[0 for j in range(10)] for i in range(10)]
  remaining = 4
  tries = 0

  def __init__(self):
    self.remaining = 4
    self.tries = 0
    #hide mugwumps
    for k in range(self.remaining):
      self.hideMugwump()

    self.playRound()

  def hideMugwump(self):
    x = random.randrange(0,9)
    y = random.randrange(0,9)
    m = self.map[x][y]
    if m != 0:
      self.hideMugwump()
    self.map[x][y] = 2

  def playRound(self):
    self.printMap()
    self.askForGuess()
    print "You have used "+str(self.tries)+" tries."
    print "There are " + str(self.remaining) + " remaining Mugwumps."
    if self.remaining > 0:
      print "....*...."
      self.playRound()
    print "Well done!"

  def printMap(self):
    print "Current grid (O-Guess and no Mugwump, M-Guess and found Mugwump):"
    print "  0 1 2 3 4 5 6 7 8 9"
    for y in range(10):
      mapString = str(y) + "|"
      for x in range(10):
        m = self.map[x][y]
        result = {
              0: lambda x: " |",
              1: lambda x: "O|",
              2: lambda x: " |", # dont show the mugwump
              3: lambda x: "M|"
```

```
            }[m](x)
            mapString += result
        print mapString

    def askForGuess(self):
        print str(self.remaining) + " Mugwumps are now in hiding. Where do you think one is?"
        x = self.askInput("What column? (0-9)")
        y = self.askInput("What row? (0-9)")
        print "...."
        self.checkResult(x,y)


    def askInput(self, s):
        y = int(raw_input(s))
        if y > 9 or y < 0:
            y = self.askInput(s)
        return y

    def checkResult(self,x,y):
        m = self.map[x][y]
        text = {
                0: lambda x: "There is no Mugwump here",
                1: lambda x: "You already looked there!",
                2: lambda x: "You found a Mugwump!",
                3: lambda x: "You already found this Mugwump!"
                }[m](x)
        if m == 0 or m == 2:
            self.map[x][y] = m+1
            self.tries = self.tries + 1
        if m == 2:
            self.remaining = self.remaining -1
        print text
        if self.remaining > 0:
            self.checkNearest(x,y)

    def checkNearest(self,x,y):
        nearestX = 10
        nearestY = 10
        nearest = self.calcDist(x,y,nearestX, nearestY)
        for nx in range(10):
            for ny in range(10):
                m = self.map[nx][ny]
                if m == 2:
                    newDist = self.calcDist(x,y,nx,ny)
                    if newDist < nearest:
                        nearest = newDist
                        nearestX = nx
                        nearestY = ny
        print "The distance to nearest Mugwumps is "+str(nearest)

    def calcDist(self,x,y,nX,nY):
        distX = x - nX
        distY = y - nY
        return math.sqrt(distX*distX+distY*distY)

# Run the game
g = Game()
```

```
package code
{
	import flash.display.MovieClip;
	import flash.text.*;
	import flash.events.*;


	public class MugwumpsText extends MovieClip
	{
		var remaining:int = 4;
		var mugwumpNr:int = 4;
		var tries:int = 0;
		// Create an array,
		// 2 dimensions, x and y
		var map:Array = new Array();

		public function MugwumpsText()
		{
			trace("MugwumpsTest2");
			text1.text = "Mugwumps Game";

			fillMap();

			for (var i:int = 0; i < remaining; i++) {
				placeMugwumps();
			}

			labelInstructions.text = "Guess the coordinates of the nearest mugwump";

			printMap();

			//yCoord.addEventListener(TextEvent.TEXT_INPUT, inputEventCapture);
			checkButton.addEventListener(MouseEvent.CLICK, inputEventCapture);
		}

		public function fillMap():void
		{
			// Fill the array of 0
			for (var i:int = 0; i < 10; i++) {
				map[i] = new Array();
				for (var j:int = 0; j < 10; j++) {
					map[i].push(0);
				}
			}
		}
		public function placeMugwumps():void
		{
			var posX:int = randomNumber(0,9);
			var posY:int = randomNumber(0,9);
			if (map[posX][posY] == 0) {
				// if empty, hide mugwump
				map[posX][posY] = 2;
			}
			else
				placeMugwumps();
		}
		public function printMap():void
		{
```

```
            var val:int = 0;
            // print the array
            textMap.text = "  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 \n";
            for (var yi:int = 0; yi < map.length; yi++) {
                textMap.text = textMap.text + yi;
                for (var xi:int = 0; xi < map[yi].length; xi++) {
                    val = map[xi][yi];
                    if ((val == 0) || (val == 2))
                    {
                        textMap.text = textMap.text + " |  ";
                    }
                    else if (val == 1)
                    {
                        textMap.text = textMap.text + " | o";
                    }
                    else if (val == 3)
                    {
                        textMap.text = textMap.text + " | x";
                    }
                    else
                    // something screwy going on
                        textMap.text = textMap.text + " | ?";

                }
                textMap.text = textMap.text + "\n";
            }
        }

        public function inputEventCapture(event:MouseEvent):void
        {
            var posY:int = int(yCoord.text);
            var posX:int = int(xCoord.text);
            labelInstructions.text = "You guessed " + posX + " x " + posY +".";

            if (map[posX][posY] == 2)
            {
                labelInstructions.text = labelInstructions.text + "\nYou found a Mugwump.";
                map[posX][posY] = 3
                remaining--;
                tries++;
            }
            else if (map[posX][posY] == 0)
            {
                labelInstructions.text = labelInstructions.text + "\nThere was no Mugwump there.";
                map[posX][posY] = 1;
                tries++;
            }
            else
            {
                labelInstructions.text = labelInstructions.text + "\nYou have already guessed there";
            }
            if (remaining > 0)
            {
                checkNearest(posX,posY);
            }
            else
                labelInstructions.text = labelInstructions.text + "\nYou have found all mugwumps.";
            text1.text = "MUGWUMPS\nTries: " + tries + "\nFound: " + (mugwumpNr - remaining);
```

```
		printMap();
	}

	public function checkNearest(posX, posY):void
	{

		var nearest:int = 1000
		var distX:int = 0;
		var distY:int = 0;
		var newNear:int = 0;
		var northSouth:String = "";
		var westEast:String = "";

		for (var xi:int = 0; xi < map.length; xi++) {
			for (var yi:int = 0; yi < map[xi].length; yi++) {
				if ((map[xi][yi] == 2) && ((xi != posX) || (yi != posY)))
				{
					distX = xi - posX;
					trace(distX);
					distY = yi - posY;
					trace(distY);
					newNear =  Math.sqrt(distX*distX + distY*distY);
					if (newNear < nearest)
					{
						nearest = newNear;
						if (distX < 0)
						{
							westEast = "West";
						}
						else if (distX > 0)
						{
							westEast = "East";
						}
						else
							westEast = "";
						if (distY < 0)
						{
							northSouth = "North";
						}
						else if (distY > 0)
						{
							northSouth = "South";
						}
						else
							northSouth = "";

					}
				}
				else
					trace("ugg");
			}
		}
		labelInstructions.text = labelInstructions.text + "\nNearest is " + nearest + " squares away to
the "
			+ northSouth + westEast +".";
	}
```

```
                    /**
* Generate a random number
* @return Random Number
* @error throws Error if low or high is not provided
*/
function randomNumber(low:Number=NaN, high:Number=NaN):Number
{
 var low:Number = low;
 var high:Number = high;

 if(isNaN(low))
 {
  throw new Error("low must be defined");
 }
 if(isNaN(high))
 {
  throw new Error("high must be defined");
 }

 return Math.round(Math.random() * (high - low)) + low;
}

          }
}
```

```
// <author>Rósa Dögg Jónsdóttir</author>
// <date>2010-16-04</date>
// <summary>Edited GamePlayScreen from Screen manager</summary>

namespace MugwumpsGameXNA
{
   #region Using Statements
   using System;
   using System.Threading;
   using Microsoft.Xna.Framework;
   using Microsoft.Xna.Framework.Content;
   using Microsoft.Xna.Framework.Graphics;
   using Microsoft.Xna.Framework.Input;
   #endregion

   /// <summary>
   /// This screen implements the actual game logic.
   /// </summary>
   class GameplayScreen : GameScreen
   {
      /// <summary>
      /// Variables for the game
      /// </summary>
      #region Fields
      /// <summary>
      /// Variables that are static for the game
      /// </summary>
      private static Vector2 foundTextLocation = new Vector2(20, 550);
      private static int mapEmpty = -1; // nothing there
      private static int mapVisible = 1; // visible object
      private static int mapVisibleHidden = 2; // visible object AND hidden object
      private static int mapVisibleSearched = 3; // searched, has no hidden
      private static int mapHiddenSearched = 4; // searched, has hidden

      /// <summary>
      /// Basic content
      /// Anything to declare for the basic game
      /// </summary>
      private static int screenWidth = 800;
      private static int screenHeight = 600;
      private static Rectangle fullscreen = new Rectangle(0, 0, screenWidth, screenHeight);
      private static int objectWidth = 50; // in pixels
      private static int objectHeight = 40; // in pixels
      private static int mapWidth = 16; // nr. of objects possible in a row
      private static int mapHeight = 10; // nr. of objects possible in a colum
      private static Rectangle mapGameScreenRect = new Rectangle(
         0, 0, objectWidth * mapWidth, objectHeight * mapHeight);

      private ContentManager content;
      private SpriteFont gameFont;
      private Texture2D gameBackground;
      private string gameBackgroundPic = "MugwumpItems\\base";
      private Vector2 basicPossition = new Vector2(0, 0);

      /// <summary>
      /// Mouse
      /// Anything to track mouse state
      /// Treat mouseclick as a box to measure for intersect.
      /// </summary>
      private MouseState mousePreviousState;
      private MouseState mouseCurrentState;
      private Rectangle mouseClickRect = new Rectangle(0, 0, 1, 1);
      private Rectangle itemLocation = new Rectangle(0, 0, objectWidth, objectHeight);
      private int mouseClickGridX = -1;
      private int mouseClickGridY = -1;

      /// <summary>
      /// Chapters
      /// The game can have X nr of chapters
      /// TODO: initialize as array?
      /// </summary>
      private Chapter chapterOne;

      /// <summary>
      /// Player
      /// Anything to declare for player
      /// Texture for player is set in the player class
      /// TODO: The player class keeps track of this, not gameplayscreen
      /// TODO: multiple chapters have different top scores
      /// </summary>
      private Player player;
      ////private int nrBestTries = 0;

      /// <summary>
      /// Anything to do with Muggy the hint giver
      /// Text for muggy is set in the muggy class
      /// </summary>
      private Muggy muggy;
      private string muggySaysNow;
      private Vector2 muggyTextLocation = new Vector2(200, 460);
      private Rectangle muggyLocation = new Rectangle(20, 460, 150, 90);
      private Texture2D muggyTexture;
      private int hintDirrection = 0;

      #endregion

      /// <summary>
      /// initializing, basic
      /// </summary>
      #region Initialization

      /// <summary>
      /// Initializes a new instance of the GameplayScreen class.
      /// Constructor.
      /// From Game screen management
      /// </summary>
      public GameplayScreen()
      {
         TransitionOnTime = TimeSpan.FromSeconds(1.5);
         TransitionOffTime = TimeSpan.FromSeconds(0.5);
      }

      /// <summary>
      /// Load graphics content for the game.
      /// </summary>
      public override void LoadContent()
```

```
{
   if (this.content == null)
   {
      this.content = new ContentManager(ScreenManager.Game.Services, "Content");
   }

   this.gameFont = this.content.Load<SpriteFont>("GameMenuItems/gamefont");

   // Load what is needed for the Mugwump Game
   this.LoadMugwumpsGame();

   // once the load has finished, we use ResetElapsedTime to tell the game's
   // timing mechanism that we have just finished a very long frame, and that
   // it should not try to catch up.
   // From game screen manager
   ScreenManager.Game.ResetElapsedTime();
}

/// <summary>
/// Unload graphics content used by the game.
/// </summary>
public override void UnloadContent()
{
   this.content.Unload();
}

#endregion

/// <summary>
/// Update and drawmethods, basic
/// </summary>
#region Update and Draw

/// <summary>
/// Updates the state of the game. This method checks the GameScreen.IsActive
/// property, so the game will stop updating when the pause menu is active,
/// or if you tab away to a different application.
/// From Game Screen Manager
/// </summary>
/// <param name="gameTime"></param>
/// <param name="otherScreenHasFocus"></param>
/// <param name="coveredByOtherScreen"></param>
public override void Update(GameTime gameTime, bool otherScreenHasFocus, bool coveredByOtherScreen)
{
   base.Update(gameTime, otherScreenHasFocus, coveredByOtherScreen);

   if (IsActive)
   {
      this.UpdateMugwumpsGame(gameTime);
   }
}

/// <summary>
/// Lets the game respond to player input. Unlike the Update method,
/// this will only be called when the gameplay screen is active.
/// From Game Screen Manager + add some mouse functions
/// RDJ: Edit to take out effecting player possition
/// </summary>
/// <param name="input"></param>
public override void HandleInput(InputState input)
{
   if (input == null)
   {
      throw new ArgumentNullException("input");
   }

   // Look up inputs for the active player profile.
   int playerIndex = (int)ControllingPlayer.Value;

   KeyboardState keyboardState = input.CurrentKeyboardStates[playerIndex];
   GamePadState gamePadState = input.CurrentGamePadStates[playerIndex];

   // The game pauses either if the user presses the pause button, or if
   // they unplug the active gamepad. This requires us to keep track of
   // whether a gamepad was ever plugged in, because we don't want to pause
   // on PC if they are playing with a keyboard and have no gamepad at all!
   bool gamePadDisconnected = !gamePadState.IsConnected &&
                        input.GamePadWasConnected[playerIndex];

   if (input.IsPauseGame(ControllingPlayer) || gamePadDisconnected)
   {
      ScreenManager.AddScreen(new PauseMenuScreen(), ControllingPlayer);
   }

   // MOUSE - added by RDJ

   // The old current state is now the previous state
   this.mousePreviousState = this.mouseCurrentState;

   // and the check state is now the current state
   this.mouseCurrentState = Mouse.GetState();

   // If the user has just clicked the Left mouse button, respond
   if ((this.mousePreviousState.LeftButton == ButtonState.Released)
        && (this.mouseCurrentState.LeftButton == ButtonState.Pressed))
   {
      this.MouseClicked();
   }
}

/// <summary>
/// Draws the gameplay screen.
/// </summary>
/// <param name="gameTime"></param>
public override void Draw(GameTime gameTime)
{
   // Mugwumps game draw method
   this.DrawMugwumpsGame(gameTime);

   // If the game is transitioning on or off, fade it out to black.
   if (TransitionPosition > 0)
   {
      ScreenManager.FadeBackBufferToBlack(255 - TransitionAlpha);
   }
}
```

```
        #endregion

        /// <summary>
        /// This is the Mugwumps game code
        /// Rósa Dögg Jónsdóttir, 2010
        /// </summary>
        #region Mugwump Game Basic Methods
        /// <summary>
        ///  Called once when loading the game,
        ///  loads muggy, player and chapter
        /// </summary>
        private void LoadMugwumpsGame()
        {
            // Initialize the previous mouse state.
            // This stores the current state of the mouse
            this.mousePreviousState = Mouse.GetState();

            // make new Muggy
            this.muggy = new Muggy();
            this.muggyTexture = this.content.Load<Texture2D>("MugwumpItems\\muggy02");

            // Have muggy say starting lines
            this.muggySaysNow = this.muggy.MuggySays(0);

            // load new Player
            this.player = new Player();
            this.player.LoadContent(this.content);

            // Basic game base
            this.gameBackground = this.content.Load<Texture2D>(this.gameBackgroundPic);

            // New chapter at start of game
            // TODO: make it so that chapters are selected
            // and created one after another
            this.chapterOne = new Chapter();
            this.chapterOne.Initialize(mapWidth, mapHeight, objectWidth, objectHeight, mapEmpty, mapVisible,
mapVisibleHidden, mapVisibleSearched, mapHiddenSearched);

            // TODO: Decide if to initialize and load chapters all in one or later on in update.
            this.chapterOne.LoadContent(this.content);
        }

        /// <summary>
        ///  Update method for Mugwumps game
        ///  Updates chapter and player,
        ///  checks if to update objects.
        /// </summary>
        /// <param name="gameTime"></param>
        private void UpdateMugwumpsGame(GameTime gameTime)
        {
            // TODO: update active chapter, instead of chapterOne
            this.chapterOne.Update(gameTime);
            this.player.Update(gameTime);

            // Check if player is at an object
            // after he was moving
            // TODO: consider moving UpdateObjects to Chapter?
            if (this.player.GetDestination() && this.player.GetMoving())
            {
                this.UpdateObjects();
                this.player.SetMoving(false);
            }
        }

        /// <summary>
        /// Draw method for Mugwumps game
        /// </summary>
        /// <param name="gameTime"></param>
        private void DrawMugwumpsGame(GameTime gameTime)
        {
            // This game has a blue background colour
            // doesn't really show unless the background is missing for some reason
            ScreenManager.GraphicsDevice.Clear(
                ClearOptions.Target, Color.CornflowerBlue, 0, 0);
            SpriteBatch spriteBatch = ScreenManager.SpriteBatch;

            // This will draw the whole thing or call methods to draw
            spriteBatch.Begin();

            // base gameboard
            spriteBatch.Draw(this.gameBackground, fullscreen, Color.White);

            // Draw the chapter specific items
            // TODO: Draw active chapter
            this.chapterOne.Draw(spriteBatch, this.gameFont, foundTextLocation);

            // Muggy says
            spriteBatch.DrawString(this.gameFont, this.muggySaysNow, this.muggyTextLocation, Color.Purple);
            spriteBatch.Draw(this.muggyTexture, this.muggyLocation, Color.White);

            // Draw player sprite
            this.player.Draw(spriteBatch);

            spriteBatch.End();
        }

        #endregion

        /// <summary>
        /// This is where the game specific logic happens
        /// </summary>
        #region Mugwumps Game Specific Methods

        /// <summary>
        /// What happens when the mouse is clicked.
        /// </summary>
        private void MouseClicked()
        {
            // Check if the player is already moving
            // if he's not moving, accept the click
            if (this.player.GetMoving())
            {
                this.muggySaysNow = "We are on our way!";
            }
            else
            {
                // take note where it was clicked
                this.mouseClickRect.X = this.mouseCurrentState.X;
                this.mouseClickRect.Y = this.mouseCurrentState.Y;

                // check where the mouseclick was in relation to the map.
                this.CheckSelection();
            }
        }

        /// <summary>
        /// Check where the mouse click was on the game board
        /// and call appropriate methods
        /// </summary>
        private void CheckSelection()
        {
            // Reset Muggy speach
            this.muggySaysNow = string.Empty;

            // TODO: active chapter, not specific
            // If found all that are hidden
            // otherwise check if it was within game board or outside
            if (this.chapterOne.GetFound() == this.chapterOne.GetHidden())
            {
                this.muggySaysNow = this.muggy.MuggySays(10);

                // TODO: offer replay or next
            }
            else if (this.mouseClickRect.Intersects(mapGameScreenRect))
            {
                // Send the player to there
                this.player.NewDestination(this.mouseClickRect.X, this.mouseClickRect.Y);

                // Assume nothing is there at start
                int locationO = mapEmpty;
                bool stop = false;
                this.mouseClickGridX = -1;

                // find the grid x and y instead of pixels
                // Loop through the map to see where we clicked.
                do
                {
                    this.mouseClickGridY = -1;
                    this.mouseClickGridX++;
                    this.itemLocation.X = this.mouseClickGridX * objectWidth;
                    do
                    {
                        this.mouseClickGridY++;
                        this.itemLocation.Y = this.mouseClickGridY * objectHeight;

                        // stop if found a match
                        stop = this.itemLocation.Intersects(this.mouseClickRect);
                    }
                    while ((this.mouseClickGridY < mapHeight) && (!stop));
                }
                while ((this.mouseClickGridX < mapWidth) && (!stop));

                // Check if anything is there
                locationO = this.chapterOne.GetMapStatus(this.mouseClickGridX, this.mouseClickGridY);
                // check if an object was clicked
                // Then Muggy will say he'll look, Player is already on his way there
                // If it wasn't an object that was clicked
                // Muggy will say there was nothing there
                if (locationO > mapEmpty)
                {
                    this.muggySaysNow = this.muggy.MuggySays(6);
                }
                else
                {
                    this.muggySaysNow = this.muggy.MuggySays(5);
                }
            }
            else
            {
                // if the selection is outside the game table
                // muggy reminds player to click on game table
                this.muggySaysNow = this.muggy.MuggySays(-1);

                // TODO: press Help, Exit etc?
            }
        }

        /// <summary>
        /// Update objects when Player arrives to it
        /// Or just update Muggy's hint if no object there
        /// </summary>
        private void UpdateObjects()
        {
            // Check if anything is there
            // TODO: active chapter
            int locationO = this.chapterOne.GetMapStatus(this.mouseClickGridX, this.mouseClickGridY);

            // check if an object is there
            // if there was, update hint and object
            // If it wasn't an object that was clicked
            // update hint
            if (locationO > mapEmpty)
            {
                int objectStatus = this.chapterOne.CheckNalterObjectStatus(this.mouseClickGridX, this.mouseClickGridY);
                this.muggySaysNow = this.muggy.MuggySays(objectStatus);
                double distance = this.chapterOne.FindDistance(this.mouseClickGridX, this.mouseClickGridY);
                this.hintDirrection = this.chapterOne.GetHintDirrection();
                this.muggySaysNow = this.muggySaysNow + this.muggy.MuggySays(distance, this.hintDirrection);
            }
            else
            {
                double distance = this.chapterOne.FindDistance(this.mouseClickGridX, this.mouseClickGridY);
                this.hintDirrection = this.chapterOne.GetHintDirrection();
                this.muggySaysNow = this.muggy.MuggySays(7) + this.muggy.MuggySays(distance, this.hintDirrection);
            }
        }

        #endregion
    }
}
```

```
// <author>Rósa Dögg Jónsdóttir</author>
// <date>2010-16-04</date>
// <summary>Chapter Class / Level of Mugwumps game</summary>

/*
 * All code is done by RDJ
 * Unless otherwise noted in the comment
 * by '-- CREATOR'
 */

namespace MugwumpsGameXNA
{
   using System;
   using System.Collections.Generic;
   using System.Linq;
   using System.Text;

   // add things for the objects
   // - XNAdev
   using Microsoft.Xna.Framework;
   using Microsoft.Xna.Framework.Content;
   using Microsoft.Xna.Framework.Graphics;
   using Microsoft.Xna.Framework.Input;

   /// <summary>
   /// Chapters are like levels of the game
   /// Are initiated by gameplayscreen
   /// </summary>
   class Chapter
   {
      /// <summary>
      /// GAME BOARD
      /// Anything to do with the game board of this chapter
      /// Last edited: RDJ 23.01.10
      /// </summary>
      #region Variables

      // Dirrections, for hint
      private static int north = 10;
      private static int south = 20;
      private static int west = 3;
      private static int east = 4;

      /* Graphics of GameBoard
       */
      private Texture2D gameBoardTexture;
      private Rectangle gameBoardPosition;
      private ContentManager content;

      /* Objects
       * an array for the objects on the field
       * This is all objects, both visible and
       * containing hidden.
       */
      private Items[] objectsInField;

      /* VARIABLES
       *
```

```
         * Any variables to set in the beginning
         * Last edited: RDJ 23.01.10
         */

        // number of objects - change later
        private int numberOfObjects = 13;

        // hidden objects
        // remember to check for nrHidden =< nrObjects
        // and stop if it is false
        private int numberOfHidden = 3;

        // Map to keep track of objects
        private int[,] map;

        // random number generator
        private Random randomNumber = new Random();

        // Hint - dirrection
        private int hintDirrection = 0;

        /* VARIABLES FROM BASIC GAME
         *
         * Variables that are in the basic game
         * get with Initialize()
         */

        // width and height of the boxes in pixels
        private int objectWidth;
        private int objectHeight;

        // width and height of map in boxes
        private int mapWidth;
        private int mapHeight;

        // used for object status
        private int mapEmpty; // nothing there
        private int mapVisible; // visible object
        private int mapVisibleHidden; // visible object AND hidden object
        private int mapVisibleSearched; // searched, has no hidden
        private int mapHiddenSearched; // searched, has hidde

        /* GRAPHICS
         *
         * Strings with location of graphics
         * Green - visible object, unsearched
         * Blue - visible object with no hidden, searched
         * Yellow - visible object with hidden, found
         * Red - Player Sprite
         */
        private string spriteGreen = "MugwumpItems\\bush01";
        private string spriteBlue = "MugwumpItems\\rocks01";
        private string spriteYellow = "MugwumpItems\\bush05";
        private string spriteRed = "MugwumpItems\\player01";
        private string gameBoardPic = "MugwumpItems\\background";

        /* KEEP TRACK
         *
```

```
 * Anything that needs keeping track of
 * Last edited: RDJ 23.01.10
 */

// to keep trach of how many found
private int numberFound = 0;
private int numberTries = 0;
private string spriteBatchText = string.Empty;

#endregion

#region Basic Methods

/// <summary>
/// Initialize the chapter, take in variables from the gameplayscreen
/// </summary>
/// <param name="mapWidth"></param>
/// <param name="mapHeight"></param>
/// <param name="objectWidth"></param>
/// <param name="objectHeight"></param>
/// <param name="mapEmpty"></param>
/// <param name="mapVisible"></param>
/// <param name="mapVisibleHidden"></param>
/// <param name="mapVisibleSearched"></param>
/// <param name="mapHiddenSearched"></param>
public void Initialize(
    int mapWidth,
    int mapHeight,
    int objectWidth,
    int objectHeight,
    int mapEmpty,
    int mapVisible,
    int mapVisibleHidden,
    int mapVisibleSearched,
    int mapHiddenSearched)
{
    this.mapWidth = mapWidth;
    this.mapHeight = mapHeight;
    this.objectWidth = objectWidth;
    this.objectHeight = objectHeight;
    this.mapEmpty = mapEmpty;
    this.mapVisible = mapVisible;
    this.mapVisibleHidden = mapVisibleHidden;
    this.mapVisibleSearched = mapVisibleSearched;
    this.mapHiddenSearched = mapHiddenSearched;

    // Initialize the chapter. Do all the basic starting things here.
    this.InitializeChapter();
}

/// <summary>
/// Load the chapter
/// Call method to load objects.
/// </summary>
/// <param name="content"></param>
public void LoadContent(ContentManager content)
{
    this.content = content;
    this.gameBoardTexture = this.content.Load<Texture2D>(this.gameBoardPic);

    // load the "items" - Possition items
    foreach (Items item in this.objectsInField)
    {
        this.LoadObjects(item);
    }
}

/// <summary>
/// Update the chapter
/// TODO: Move from gameplayscreen.update?
/// </summary>
/// <param name="gameTime"></param>
public void Update(GameTime gameTime)
{
    // all update is currently in gameplayscreen.cs
    // TODO:Move to chapter
}

/// <summary>
/// draw the chapter specific graphic
/// </summary>
/// <param name="theSpriteBatch"></param>
/// <param name="gameFont"></param>
/// <param name="foundTextLocation"></param>
public void Draw(SpriteBatch theSpriteBatch, SpriteFont gameFont, Vector2 foundTextLocation)
{
    // Draw the objects
    theSpriteBatch.Draw(this.gameBoardTexture, this.gameBoardPosition, Color.White);
    foreach (Items item in this.objectsInField)
    {
        item.Draw(theSpriteBatch);
    }

    // List found and hidden items
    this.spriteBatchText = "Found: " + this.GetFound() + "/ " + this.GetHidden() + "\nTries: " + this.GetTries();
    theSpriteBatch.DrawString(gameFont, this.spriteBatchText, foundTextLocation, Color.Black);
}

#endregion

#region GETTERS AND SETTERS

/// <summary>
/// Return number of found Mugwumps
/// </summary>
/// <returns>numberFound</returns>
public int GetFound()
{
    return this.numberFound;
}

/// <summary>
/// Set number of found Mugwumps
/// </summary>
/// <param name="numberFound"></param>
```

```
public void SetFount(int numberFound)
{
   this.numberFound = numberFound;
}

/// <summary>
/// Return number of hidden Mugwumps
/// </summary>
/// <returns>nrHidden</returns>
public int GetHidden()
{
   return this.numberOfHidden;
}

/// <summary>
/// Set how often player has guessed
/// </summary>
/// <param name="numberTries"></param>
public void SetTries(int numberTries)
{
   this.numberTries = numberTries;
}

/// <summary>
/// Return number of tries
/// </summary>
/// <returns>numberTries</returns>
public int GetTries()
{
   return this.numberTries;
}

/// <summary>
/// Returns hint dirrection
/// </summary>
/// <returns>hintDirrection</returns>
public int GetHintDirrection()
{
   return this.hintDirrection;
}

/// <summary>
/// Returns what is on the map at (X,Y)
/// </summary>
/// <param name="locationX"></param>
/// <param name="locationY"></param>
/// <returns>map[locationX,locationY]</returns>
public int GetMapStatus(int locationX, int locationY)
{
   return this.map[locationX, locationY];
}

#endregion

#region Functions

/// <summary>
/// Check and alter object status based on (x,y) location
/// called by gameplayscreen
/// </summary>
/// <param name="locationX"></param>
/// <param name="locationY"></param>
/// <returns>map status at location</returns>
public int CheckNalterObjectStatus(int locationX, int locationY)
{
   // find out what object was clicked
   int locationO = this.GetMapStatus(locationX, locationY);
   int objectStatus = this.objectsInField[locationO].GetStatus();

   // not searched, has no hidden
   if (objectStatus == this.mapVisible)
   {
      this.objectsInField[locationO].LoadContent(this.content, this.spriteBlue);
      this.objectsInField[locationO].SetStatus(this.mapVisibleSearched);

      return this.mapVisible;
   }
   else if (objectStatus == this.mapVisibleHidden)
   {
      // not searched, has hidden)
      this.objectsInField[locationO].LoadContent(this.content, this.spriteYellow);
      this.objectsInField[locationO].SetStatus(this.mapHiddenSearched);

      // count found
      this.numberFound++;

      return this.mapVisibleHidden;
   }

   return objectStatus;
}

/// <summary>
/// Find distance from location to nearest hidden Mugwump
/// </summary>
/// <param name="locationX">x-coordinate</param>
/// <param name="locationY">y-coordinate</param>
/// <returns>distance</returns>
public double FindDistance(int locationX, int locationY)
{
   // every time you check for distance, you 'try'
   this.numberTries++;

   double distanceNear = this.mapEmpty;
   double distanceNew = distanceNear;
   int distanceX = 0;
   int distanceY = 0;
   foreach (Items item in this.objectsInField)
   {
      if (item.GetStatus() == this.mapVisibleHidden)
      {
         distanceX = locationX - item.GetMapX();
         distanceY = locationY - item.GetMapY();

         // find out if it has hidden
         // if it has hidden check if it is nearer than the last one
```

```
                // if it is nearer then mark as nearest
                distanceNew = Math.Sqrt((distanceX * distanceX) + (distanceY * distanceY));

                if ((distanceNear == this.mapEmpty) || (distanceNew < distanceNear))
                {
                    distanceNear = distanceNew;
                    this.hintDirrection = 0;

                    // dirrection
                    if (distanceY > 0)
                    {
                        this.hintDirrection = north;
                    }
                    else if (distanceY < 0)
                    {
                        this.hintDirrection = south;
                    }

                    if (distanceX < 0)
                    {
                        this.hintDirrection = this.hintDirrection + east;
                    }
                    else if (distanceX > 0)
                    {
                        this.hintDirrection = this.hintDirrection + west;
                    }
                }
            }
        }

        return distanceNear;
    }

    #endregion

    #region Basic Chapter Initialization

    /// <summary>
    /// Initialize the chapter objects
    /// </summary>
    private void InitializeChapter()
    {
        this.map = new int[this.mapWidth, this.mapHeight];

        // the map starts 'empty'
        for (int index = 0; index < this.mapWidth; index++)
        {
            for (int index2 = 0; index2 < this.mapHeight; index2++)
            {
                this.map[index, index2] = this.mapEmpty;
            }
        }

        // array of the objects on the game board
        this.objectsInField = new Items[this.numberOfObjects];
        for (int index = 0; index < this.numberOfObjects; index++)
        {
            this.objectsInField[index] = new Items();
            // Put the object on the map
            this.PlaceObject(this.objectsInField[index], index);
        }

        // hide the mugwumps / hidden objects
        this.HideMugwumps();

        // set the gameboard
        this.gameBoardPosition = new Rectangle(0, 0, this.mapWidth * this.objectWidth, this.mapHeight *
this.objectHeight);
    }

    /// <summary>
    /// find an empty spot on the map to place item
    /// </summary>
    /// <returns>return a free hiding spot</returns>
    private int[] FindSpot()
    {
        int[] spot = new int[2];
        int x = this.randomNumber.Next(this.mapWidth);
        int y = this.randomNumber.Next(this.mapHeight);

        // if the map spot is not empty
        // find a new spot
        if (this.map[x, y] > this.mapEmpty)
        {
            spot = this.FindSpot();
        }
        else
        {
            spot[0] = x;
            spot[1] = y;
        }

        return spot;
    }

    /// <summary>
    /// Put the object on the map
    /// </summary>
    /// <param name="item"></param>
    /// <param name="itemNR"></param>
    private void PlaceObject(Items item, int itemNR)
    {
        // find a spot to put the object
        int[] spot = new int[2]; // a spot needs an x and y coordinate
        spot = this.FindSpot();
        int x = spot[0];
        int y = spot[1];

        // possition on the map
        item.SetPosition(x, y, this.objectWidth, this.objectHeight);

        // mark the item as a visible item
        item.SetStatus(this.mapVisible);

        // Note the item's number on the map location
```

```
            this.map[x, y] = itemNR;
        }

        /// <summary>
        /// Find a place to hide the mugwump
        /// Must not contain a mugwump already
        /// </summary>
        /// <returns>Return spot for hiding place</returns>
        private int FindHidingplace()
        {
            int spot = this.randomNumber.Next(this.numberOfObjects);

            // if the object has a hidden
            // find another object
            if (this.objectsInField[spot].GetStatus() > this.mapVisible)
            {
                spot = this.FindHidingplace();
            }

            return spot;
        }

        /// <summary>
        /// hide the mugwumps in an empty object
        /// </summary>
        private void HideMugwumps()
        {
            int spot = 0; // The nr of the object it will hide in
            for (int index = 0; index < this.numberOfHidden; index++)
            {
                spot = this.FindHidingplace();

                // Set the object as a visible with hidden
                this.objectsInField[spot].SetStatus(this.mapVisibleHidden);
            }
        }

        /// <summary>
        /// Loads sprites for objects
        /// Display spriteRed if something odd is going on
        /// TODO: change this method to generate 'random' looks
        /// </summary>
        /// <param name="item"></param>
        private void LoadObjects(Items item)
        {
            if (item.GetStatus() > this.mapEmpty)
            {
                item.LoadContent(this.content, this.spriteGreen);
            }
            else
            {
                item.LoadContent(this.content, this.spriteRed);
            }
        }
        #endregion
    }
}
```

```
// <copyright file="Player.cs" >
// Copyright (c) 2010 All Right Reserved
// </copyright>
// <author>Rósa Dögg Jónsdóttir</author>
// <date>2010-16-04</date>
// <summary>Player class for Mugwumps</summary>

namespace MugwumpsGameXNA
{
   using System;
   using System.Collections.Generic;
   using System.Linq;
   using System.Text;

   // add things for the objects
   // - XNAdev
   using Microsoft.Xna.Framework;
   using Microsoft.Xna.Framework.Content;
   using Microsoft.Xna.Framework.Graphics;

   /// <summary>
   /// player of mugwump
   /// TODO: keep track of best scores and other progress
   /// </summary>
   class Player
   {
      #region Variables

      // Size of the player sprite
      private static int objectWidth = 55;
      private static int objectHeight = 50;

      // Keep track of position to
      // have the sprite move a bit at a time
      // set the player outside the game board at start, but visible
      private Vector2 position = new Vector2(530, 530);
      private float distX = 0;
      private float distY = 0;
      private int dirrX = 0;
      private int dirrY = 0;
      private int theSpeed = 50;

      // destination possition of the sprite
      private Vector2 destPosition = new Vector2(50, 50);
      private bool arrivedDestination = false;
      private bool moving = false;

      // TODO : Animation using more textures
      private Texture2D mapSpriteTexture;
      private Rectangle playerBox = new Rectangle(0, 0, objectWidth, objectHeight);
      private string texturePlayer = "MugwumpItems\\player01";

      #endregion

      #region Basic Methods

      /// <summary>
      /// Load the texture for the sprite using the Content Pipeline
      /// </summary>
      /// <param name="theContentManager"></param>
      public void LoadContent(ContentManager theContentManager)
      {
         this.mapSpriteTexture = theContentManager.Load<Texture2D>(this.texturePlayer);
      }

      /// <summary>
      /// calls MovePlayer to update player location
      /// </summary>
      /// <param name="gameTime"></param>
      public void Update(GameTime gameTime)
      {
         this.MovePlayer(gameTime);
      }

      /// <summary>
      /// Draw the sprite to the screen
      /// </summary>
      /// <param name="theSpriteBatch"></param>
      public void Draw(SpriteBatch theSpriteBatch)
      {
         this.playerBox.X = (int)this.position.X;
         this.playerBox.Y = (int)this.position.Y;
         theSpriteBatch.Draw(this.mapSpriteTexture, this.playerBox, Color.White);
      }
      #endregion

      #region GETTERS AND SETTERS

      /// <summary>
      /// return the player sprite texture
      /// </summary>
      /// <returns></returns>
      public string GetTexture()
      {
         return this.texturePlayer;
      }

      /// <summary>
      /// Set a new texture for the player sprite
      /// </summary>
      /// <param name="texturePlayer"></param>
      public void SetTexture(string texturePlayer)
      {
         this.texturePlayer = texturePlayer;
      }

      /// <summary>
      /// Return wether the player is at a destination or not.
      /// </summary>
      /// <returns></returns>
      public bool GetDestination()
      {
         return this.arrivedDestination;
      }

      /// <summary>
      /// send the player to moving
      /// </summary>
      /// <param name="moving"></param>
      public void SetMoving(bool moving)
      {
         this.moving = moving;
      }

      /// <summary>
      /// check if the player is moving
      /// </summary>
      /// <returns></returns>
      public bool GetMoving()
      {
         return this.moving;
      }

      /// <summary>
      /// check the possition of the player sprite
      /// </summary>
      /// <returns></returns>
      public Vector2 GetPosition()
      {
         return this.position;
      }
      #endregion

      #region Functions
      /// <summary>
      /// set a new destination
      /// </summary>
      /// <param name="mouseX"></param>
      /// <param name="mouseY"></param>
      public void NewDestination(int mouseX, int mouseY)
      {
         // destination of center of the sprite
         // is to the mouseclick
         this.destPosition.X = mouseX - (objectWidth / 2);
         this.destPosition.Y = mouseY - (objectHeight / 2);
         this.arrivedDestination = false;
         this.moving = true;
         this.CalcDistance();
      }

      /// <summary>
      /// Move the player if he is not at his destination
      /// TODO: check this, seems to be some glitch happening sometimes
      /// with not moving the whole distance.
      /// </summary>
      /// <param name="gameTime"></param>
      private void MovePlayer(GameTime gameTime)
      {
         // if the X of destination and current position is not the same, move a bit closer
         if (this.distX > 0 && this.moving)
         {
            this.position.X += this.dirrX * this.theSpeed * (float)gameTime.ElapsedGameTime.TotalSeconds;
            this.distX--;
         }
```

```
            // if the Y is no the same move a bit closer
            if ((this.distY > 0) && this.moving)
            {
                this.position.Y += this.dirrY * this.theSpeed * (float)gameTime.ElapsedGameTime.TotalSeconds;
                this.distY--;
            }

            if ((this.distY < 1) && (this.distX < 1) && this.moving)
            {
                this.arrivedDestination = true;
            }
        }

        /// <summary>
        /// Calculate the distance from current to new destination
        /// </summary>
        private void CalcDistance()
        {
            this.distX = Math.Abs(this.destPosition.X - this.position.X);
            this.distY = Math.Abs(this.destPosition.Y - this.position.Y);
            if (this.destPosition.X < this.position.X)
            {
                this.dirrX = -1;
            }
            else
            {
                this.dirrX = 1;
            }

            if (this.destPosition.Y < this.position.Y)
            {
                this.dirrY = -1;
            }
            else
            {
                this.dirrY = 1;
            }
        }

        #endregion
    }
}

// <copyright file="Muggy.cs" >
// Copyright (c) 2010 All Right Reserved
// </copyright>
// <author>Rósa Dögg Jónsdóttir</author>
// <date>2010-16-04</date>
// <summary>Muggy the hintgiver for Mugwumps</summary>

namespace MugwumpsGameXNA
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
```

```
// add things for the objects - XNAdev
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;

/// <summary>
/// Muggy the hint giver, his words
/// </summary>
class Muggy
{
   #region Variables
   // random number generator
   Random randomNumber = new Random();
   #endregion

   #region Functions
   /// <summary>
   /// Returns what Muggy should say
   /// All strings of what Muggy can say are in this
   /// and muggySays(double distance)
   /// TODO: add randomness and intelligence?
   /// </summary>
   /// <param name="selection"></param>
   /// <returns></returns>
   public string MuggySays(int selection)
   {
      // if user clicked outside the map
      if (selection == -1)
      {
         return "You have to select a place on the map.";
      }

      // if user clicked an object that does not contain a mugwump
      if (selection == 1)
      {
         return "This bush is not hiding a Mugwump.\n";
      }

      // if user clicked an object that does contain a mugwump
      if (selection == 2)
      {
         return "You have found a Mugwump!\n";
      }

      // if already searched there
      if (selection == 3)
      {
         if (this.randomNumber.Next(20) == 0)
         {
            return "Hey! Try looking in one of the other bushes\n";
         }
         else
         {
            return "You have already searched there.\n";
         }
      }

      // if selection was a mugwump found
```

```
        if (selection == 4)
        {
            if (this.randomNumber.Next(20) == 0)
            {
                return "He says it tickles!\n";
            }
            else
            {
                return "You have already found this Mugwump.\n";
            }
        }

        // if selection is not object
        if (selection == 5)
        {
            if (this.randomNumber.Next(20) == 0)
            {
                return "You should be trying to look in the bushes\n but lets go and look there...";
            }
            else
            {
                return "There is nothing there\n but let's try going there and looking...";
            }
        }

        // if selection is not searched object so heading there
        if (selection == 6)
        {
            return "Alright, lets search there!";
        }

        // when arriving at empty spot
        if (selection == 7)
        {
            return "Alright, lets see what the magic stone says!";
        }

        if (selection == 10)
        {
            if (this.randomNumber.Next(2) == 0)
            {
                return "All Mugwumps have been found,\nwell done!\n Press ESC to quit";
            }
            else
            {
                return "You have found all Mugwumps.";
            }
        }

        return "Help me find the hidden Mugwumps,\n they are hiding behind a green blob\n\nGuess by clicking on a
blob,\n\nI will give you hints!";
    }

    /// <summary>
    /// have muggy say where nearest mugwump is
    /// return distance and dirrection
    /// unless the distance is nonesense
    /// </summary>
/// <param name="distance"></param>
/// <param name="hintDirrection"></param>
/// <returns></returns>
public string MuggySays(double distance, int hintDirrection)
{
   // if the distance is real
   if (distance > -1)
   {
      return this.MuggySaysDirrection(distance, hintDirrection);
   }
   else
   {
      return this.MuggySays(10);
   }
}

/// <summary>
/// In which dirrection the closest mugwump is
/// and how close
/// </summary>
/// <param name="distance"></param>
/// <param name="dirrection"></param>
/// <returns></returns>
private string MuggySaysDirrection(double distance, int dirrection)
{
   string closeness = "somewhere away from";
   string final = string.Empty;
   if (distance < 2)
   {
      closeness = "very close to";
   }
   else if (distance < 3)
   {
      closeness = "close to";
   }
   else if (distance < 5)
   {
      closeness = "not far away from";
   }
   else if (distance < 105)
   {
      closeness = "far away from";
   }
   else
   {
      closeness = "very far away from";
   }

   final = "\nThe next mugwump is " + closeness + " \nyou, to ";

   if (dirrection == 10)
   {
      final = final + "the north";
   }
   else if (dirrection == 20)
   {
      final = final + "the south";
   }
```

```
            else if (dirrection == 3)
            {
                final = final + "the west";
            }
            else if (dirrection == 4)
            {
                final = final + "the east";
            }
            else if (dirrection == 13)
            {
                final = final + "the north west";
            }
            else if (dirrection == 14)
            {
                final = final + "the north east";
            }
            else if (dirrection == 23)
            {
                final = final + "the south west";
            }
            else if (dirrection == 24)
            {
                final = final + "the south east";
            }
            else
            {
                final = final + "an unknown dirrection";
            }

            return final;
        }
        #endregion
    }
}


// <author>Rósa Dögg Jónsdóttir</author>
// <date>2010-16-04</date>
// <summary>Visible Objects for Mugwumps</summary>

namespace MugwumpsGameXNA
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    // add things for the objects
    // - XNAdev
    using Microsoft.Xna.Framework;
    using Microsoft.Xna.Framework.Content;
    using Microsoft.Xna.Framework.Graphics;

    /// <summary>
    /// Items that are visible
    /// </summary>
    class Items
    {
```

```
#region Variables

// The current position of the Sprite
// The texture object used when drawing the sprite
private Texture2D mapSpriteTexture;
private Rectangle objectBox;

// possition in map
private int mapX;
private int mapY;

// status:
// 0 = unknown
// 1 = not searched, has no hidden
// 2 = not searched, has hidden
// 3 = searched, has no hidden
// 4 = searched, has hidden
private int status;
#endregion

#region Basic Methods
/// <summary>
/// Load the texture for the sprite using the Content Pipeline
/// </summary>
/// <param name="theContentManager"></param>
/// <param name="theAssetName"></param>
public void LoadContent(ContentManager theContentManager, string theAssetName)
{
   this.mapSpriteTexture = theContentManager.Load<Texture2D>(theAssetName);
}

/// <summary>
/// Draw the sprite to the screen
/// </summary>
/// <param name="theSpriteBatch"></param>
public void Draw(SpriteBatch theSpriteBatch)
{
   theSpriteBatch.Draw(this.mapSpriteTexture, this.objectBox, Color.White);
}

#endregion

#region GETTERS AND SETTERS

/// <summary>
/// set possition of object
/// </summary>
/// <param name="x"></param>
/// <param name="y"></param>
/// <param name="objectWidth"></param>
/// <param name="objectHeight"></param>
public void SetPosition(int x, int y, int objectWidth, int objectHeight)
{
   this.mapX = x;
   this.mapY = y;
   this.objectBox = new Rectangle(x * objectWidth, y * objectHeight, objectWidth, objectHeight);
}
```

```
/// <summary>
/// return object possition
/// </summary>
/// <returns></returns>
public Rectangle GetPosition()
{
   return this.objectBox;
}

/// <summary>
/// Set texture for sprite
/// </summary>
/// <param name="mapSpriteTexture"></param>
public void SetTexture(Texture2D mapSpriteTexture)
{
   this.mapSpriteTexture = mapSpriteTexture;
}

/// <summary>
/// Return texture for sprite
/// </summary>
/// <returns></returns>
public Texture2D GetTexture()
{
   return this.mapSpriteTexture;
}

/// <summary>
/// Set Map X-coordination
/// </summary>
/// <param name="mapX"></param>
public void SetMapX(int mapX)
{
   this.mapX = mapX;
}

/// <summary>
/// Return X-coordination
/// </summary>
/// <returns></returns>
public int GetMapX()
{
   return this.mapX;
}

/// <summary>
/// Set Y-coordination
/// </summary>
/// <param name="mapY"></param>
public void SetMapY(int mapY)
{
   this.mapY = mapY;
}

/// <summary>
/// Return Y-coordination
/// </summary>
/// <returns></returns>
```

```
        public int GetMapY()
        {
            return this.mapY;
        }

        /// <summary>
        /// Set status of object
        /// TODO: limit to legal value
        /// </summary>
        /// <param name="status"></param>
        public void SetStatus(int status)
        {
            this.status = status;
        }

        /// <summary>
        /// return status of object
        /// </summary>
        /// <returns></returns>
        public int GetStatus()
        {
            return this.status;
        }

        #endregion
    }
}
```

```
"""

Mugwumps Graphic game

Uses chess tutorial from Panda3D as base

Rosa Dogg Jonsdottir

"""

# Imports for basic game
import random
import math
import sys

# Imports for Panda3D basic
import direct.directbase.DirectStart
from direct.gui.OnscreenText import OnscreenText
from pandac.PandaModules import PandaNode
from pandac.PandaModules import NodePath
from pandac.PandaModules import Camera
from pandac.PandaModules import TextNode
from direct.gui.DirectGui import *

# Imports for the graphic chessboard
from panda3d.core import CollisionTraverser
from panda3d.core import CollisionNode
from panda3d.core import CollisionHandlerQueue
from panda3d.core import CollisionRay
from panda3d.core import AmbientLight
from panda3d.core import DirectionalLight
from panda3d.core import LightAttrib
from panda3d.core import Point3
from panda3d.core import Vec3
from panda3d.core import Vec4
from panda3d.core import BitMask32
from direct.showbase.DirectObject import DirectObject
from direct.task.Task import Task
import MugwumpsGraphic

# Define some constants for the colors
BLACK = Vec4(0,0,0,1)
WHITE = Vec4(1,1,1,1)
GREEN = Vec4(0,1,0,1) #R,G,B,A - RDJ
YELLOW = Vec4(1,1,0,1)
HIGHLIGHT = Vec4(0,.5,0,1)
PIECEBLACK = Vec4(.15, .15, .15, 1)

#Constant for the grid - RDJ
MAPY = 8
MAPX = 8

#Helper functions for the map


def PointAtZ(z, point, vec):
    """
    This function, given a line (vector plus origin point) and a desired z value,
    will give us the point on the line where the desired z value is what we want.
    This is how we know where to position an object in 3D space based on a 2D mouse
    position. It also assumes that we are dragging in the XY plane.
    This is derived from the mathmatical of a plane, solved for a given point
    -- chess tutorial

    """
    return point + vec * ((z-point.getZ()) / vec.getZ())


def SquarePos(i):
    """

    A little function for getting the proper position for a given square
    -- chess tutorial

    """
    return Point3((i%MAPX) - 3.5, int(i/MAPY) - 3.5, 0)



def SquareColor(i):
    """

    Helper function for determining whether a square should be white or black
    The modulo operations (%) generate the every-other pattern of a chess-board
    -- chess tutorial

    """
    # RDJ: always green in mugwumps
    # TODO: place objects, make squares different color if has object?

    return GREEN


class Game(DirectObject):
    """

    The Mugwump game class
    Initiates the game.

    """
    # TODO: Initiates chapter, offers replay

    nrItems = 10        # The items Mugwumps can hide in
    mugwumpsTotal = 4   # How many Mugwumps
    remaining = mugwumpsTotal
    tries = 0           # Keep count of how many tries the player has used
    # initialize map, status:
        # 0 - none, starting status
        # 1 - searched, no mugwump
        # 2 - mugwump in hiding
        # 3 - searched, mugwump found
        # 5 - unsearched item
    map = [[0 for j in range(MAPY)] for i in range(MAPX)]
    # Squares form the map
    squares = [None for i in range(MAPY*MAPX)]
```

```
mapLine = [-1 for i in range(MAPY*MAPX)]
# Pieces are the visible objects
items = [None for i in range(nrItems)]
mugwumps = [None for i in range(remaining)]

# Direction, initiate with 'some'
# TODO: all mugwump comments should be in separate function?
northSouth = "some"
eastWest = "some"

# We will attach all of the squares to their own root. This way we can do the
# collision pass just on the squares and save the time of checking the rest
# of the scene
# -- chess tutorial
squareRoot = render.attachNewNode("squareRoot")

#TEXT
titleText = OnscreenText(text="Panda3D: Mugwumps",
                         style=1, fg=(1,1,1,1),
                         pos=(0.8,-0.95), scale = .07)
escapeText = OnscreenText(text="ESC: Quit",
                         style=1, fg=(1,1,1,1), pos=(-1.3, 0.95),
                         align=TextNode.ALeft, scale = .05)
mugwumpText = OnscreenText(text="Left-click to quess the location of Mugwumps\n"
                          + "You have used " + str(tries) + " tries",
                         style=1, fg=(1,1,1,1), pos=(-1.3, 0.90),
                         align=TextNode.ALeft, scale = .05)
muggyText = OnscreenText(text="Hi there!",
                         style=1, fg=(1,1,1,1), pos=(0, -.5),
                         scale = .05)

def __init__(self):
    """

    Initialize method from chess tutorial
    edited by RDJ

    """
    self.accept('escape', sys.exit)              # Escape quits
    base.disableMouse()                          # Disble mouse camera control
    camera.setPosHpr(0, -13.75, 6, 0, -25, 0)    # Set the camera
    self.setupLights()

    # Just to see the X and Y coordinates for testing
    for i in range(-10,10):
        OnscreenText(text = '.', pos = (0, i/float(10)), scale = 0, fg=(1,1,1,1))
    for i in range(-10,10):
        OnscreenText(text = '.', pos = (i/float(10), 0), scale = 0, fg=(1,1,1,1))

    # MOUSE
    # Since we are using collision detection to do picking, we set it up like
    # any other collision detection system with a traverser and a handler
    self.picker = CollisionTraverser()            # Make a traverser
    self.pq     = CollisionHandlerQueue()         # Make a handler
    # Make a collision node for our picker ray
    self.pickerNode = CollisionNode('mouseRay')
    # Attach that node to the camera since the ray will need to be positioned
    # relative to it
```

```
        self.pickerNP = camera.attachNewNode(self.pickerNode)
        # Everything to be picked will use bit 1. This way if we were doing other
        # collision we could seperate it
        self.pickerNode.setFromCollideMask(BitMask32.bit(1))
        self.pickerRay = CollisionRay()              # Make our ray
        self.pickerNode.addSolid(self.pickerRay)      # Add it to the collision node
        # Register the ray as something that can cause collisions
        self.picker.addCollider(self.pickerNP, self.pq)
        #self.picker.showCollisions(render)

        # MAP
        self.drawSquares()

        # Place items
        for k in range(self.nrItems):
            squareNr = self.placeItems(k)
            self.mapLine[squareNr] = k
            self.items[k] = Item(squareNr, WHITE)

        # Hide mugwumps
        for k in range(self.mugwumpsTotal):
            x = self.hideMugwump(k)
            self.mugwumps[k] = Mugwump(False,x)
            self.items[x].mugwumpNr = k
            #self.items[x].setColor(PIECEBLACK) #for testing

        # Display starting text
        self.writeMuggyText("Try to find the Mugwumps!")

        # This will represent the index of the currently highlighted square
        self.hiSq = False
        # This is will represent where mousebutton was pressed
        self.dragging = False

        # Start the task that handles the picking
        self.mouseTask = taskMgr.add(self.mouseTask, 'mouseTask')
        self.accept("mouse1", self.mouse1Down)       #left-click grabs a piece
        self.accept("mouse1-up", self.mouse1Up) #releasing places it

        # GAME PLAY
        # Takes place from mouse1Up

    # MOUSE
    def mouseTask(self, task):
        """

        Mouse task - listen to the mouse

        """

        # This task deals with the highlighting and dragging based on the mouse
        # First, clear the current highlight
        if self.hiSq is not False:
            self.hiSq = False

        # Check to see if we can access the mouse. We need it to do anything else
        if base.mouseWatcherNode.hasMouse():
            # get the mouse position
            mpos = base.mouseWatcherNode.getMouse()

            # Set the position of the ray based on the mouse position
            self.pickerRay.setFromLens(base.camNode, mpos.getX(), mpos.getY())

            # Do the actual collision pass (Do it only on the squares for
            # efficiency purposes)
            self.picker.traverse(self.squareRoot)
            if self.pq.getNumEntries() > 0:
                # if we have hit something, sort the hits so that the closest
                # is first, and highlight that node
                self.pq.sortEntries()
                i = int(self.pq.getEntry(0).getIntoNode().getTag('square'))
                # Set the highlight on the picked square
                # self.squares[i].setColor(HIGHLIGHT)
                self.hiSq = i

        return Task.cont

    def mouse1Down(self):
        """

        If mouse button is pressed down

        """

        # Note location when mousebutton pressed
        if (self.hiSq is not False and self.dragging is False):
            self.dragging = self.hiSq
            self.hiSq = False

    def mouse1Up(self):
        """

        When the mouse button is released

        """

        s = "Click on the map"      # Default, remind the user to click on the map
        # If we were holding down the mouse button and are on the board
        if self.remaining == 0:
            # IF all are found just stop
            s = "You have found all Mugwumps"
        elif (self.hiSq is not False and self.dragging is not False):
            # Check if we are on the same square as before
            if self.hiSq == self.dragging:
                s = self.checkSelection()
        self.writeMuggyText(s)
        self.mugwumpText.setText("Left-click to quess the location of Mugwumps\n" +
                        "You have used " + str(self.tries) + " tries")
        self.dragging = False

    # DRAW MAP
    def drawSquares(self):
        """

        Draw the map / squares
        """
        for i in range(MAPY * MAPX):
            # Load, parent, color, and position the model (a single square polygon)
            self.squares[i] = loader.loadModel("Textures/square")
            self.squares[i].reparentTo(self.squareRoot)
            self.squares[i].setPos(SquarePos(i))
            self.squares[i].setColor(SquareColor(i))

            # Set the model itself to be collide able with the ray. If this model was
            # any more complex than a single polygon, you should set up a collision
            # sphere around it instead. But for single polygons this works fine.
            self.squares[i].find("**/polygon").node().setIntoCollideMask(
              BitMask32.bit(1))
            # Set a tag on the square's node so we can look up what square this is
            # later during the collision pass
            self.squares[i].find("**/polygon").node().setTag('square', str(i))

    # MUGWUMP DEF
    def placeItems(self, itemNr):
        """

        Place visible objects on the map

        """
        x = random.randrange(0,MAPX - 1)
        y = random.randrange(0,MAPX - 1)
        squareNr = y * 8 + x
        m = self.mapLine[squareNr]
        if m != -1:
        # If the square is occupied, try again
            squareNr = self.placeItems(itemNr)
        return squareNr

    def hideMugwump(self, mugwumpNr):
        """

        Hide mugwumps inside visible objects

        """

        x = random.randrange(0,self.nrItems - 1)
        m = self.items[x].mugwumpNr
        if m != -1:
        # if the item has a mugwump already
            x = self.hideMugwump(mugwumpNr)
        return x

    def writeMuggyText(self, s):
        """

        have Muggy say s

        """
        self.muggyText.setText(s)

    def calcDistance(self, nearLoc, nearest):
        """
        Find the nearest hidden Mugwump

        """
        x = self.dragging % 8
        y = self.dragging / 8
        nX = nearLoc % 8
        nY = nearLoc / 8
        distX = x - nX
        distY = y - nY
        newDist = math.sqrt(distX*distX+distY*distY)
        if newDist < nearest:
            if distX > 0:
                self.eastWest = "West"
            elif distX < 0:
                self.eastWest = "East"
            else:
                self.eastWest = ""
            if distY > 0:
                # north and south is confused because of camera angle
                self.northSouth = "South"
            elif distY < 0:
                self.northSouth = "North"
            else:
                self.northSouth = ""
            return newDist
        else:
            return nearest

    def textDistance(self):
        """

        Say how far away the hidden mugwump is

        """
        s = "some"
        # Start somewhere far away
        nearest = MAPY * MAPX * 1000
        for i in range(self.mugwumpsTotal):
            m = self.mugwumps[i]
            if m.isFound is False:
                loc = self.items[m.location].location
                nearest = self.calcDistance(loc, nearest)

        if nearest < 2:
            s = "very close"
        elif nearest < 3:
            s = "close"
        elif nearest < 5:
            s = "not very far"
        else:
            s = "far"
        return "\nThe nearest Mugwump is " + s + " to the " + self.northSouth + self.eastWest

    def checkSelection(self):
        """

        When the player has selected a square check what is there.
        """
        s = "Guess!"
        if self.squares[self.dragging].getColor() != GREEN:
            # If already searched
            s = "You searched there already"
        elif self.mapLine[self.dragging] > -1:
        # If there is a piece on the square
            self.tries = self.tries + 1
            iNr = self.mapLine[self.dragging]
            if self.items[iNr].searched is not True:
                self.items[iNr].searched = True
                if self.items[iNr].mugwumpNr > -1:
                # if it has hidden
                    mNr = self.items[iNr].mugwumpNr
                    #if it has hidden, set found
                    self.squares[self.dragging].setColor(HIGHLIGHT)
                    self.remaining = self.remaining -1
                    s = "Well done you have found a Mugwump!"
                    self.mugwumps[mNr].isFound = True
                else:
                    s = "There was no Mugwump there"
                    self.squares[self.dragging].setColor(YELLOW)
        else:
            s = "There was no Mugwump there"
            self.squares[self.dragging].setColor(YELLOW)
        if self.remaining != 0:
            s = s + self.textDistance()
        else:
            s = s + "\nYou have found all Mugwumps"
        return s

    def setupLights(self):
        """

        This function sets up some default lighting
        -- Chess tutorial

        """
        ambientLight = AmbientLight( "ambientLight" )
        ambientLight.setColor( Vec4(.8, .8, .8, 1) )
        directionalLight = DirectionalLight( "directionalLight" )
        directionalLight.setDirection( Vec3( 0, 45, -45 ) )
        directionalLight.setColor( Vec4( 0.2, 0.2, 0.2, 1 ) )
        render.setLight(render.attachNewNode( directionalLight ) )
        render.setLight(render.attachNewNode( ambientLight ) )


class Mugwump:
    """

    Mugwump class, the hidden mugwumps

    """
    def __init__(self,isFound, location):
        self.isFound = isFound
        self.location = location
```

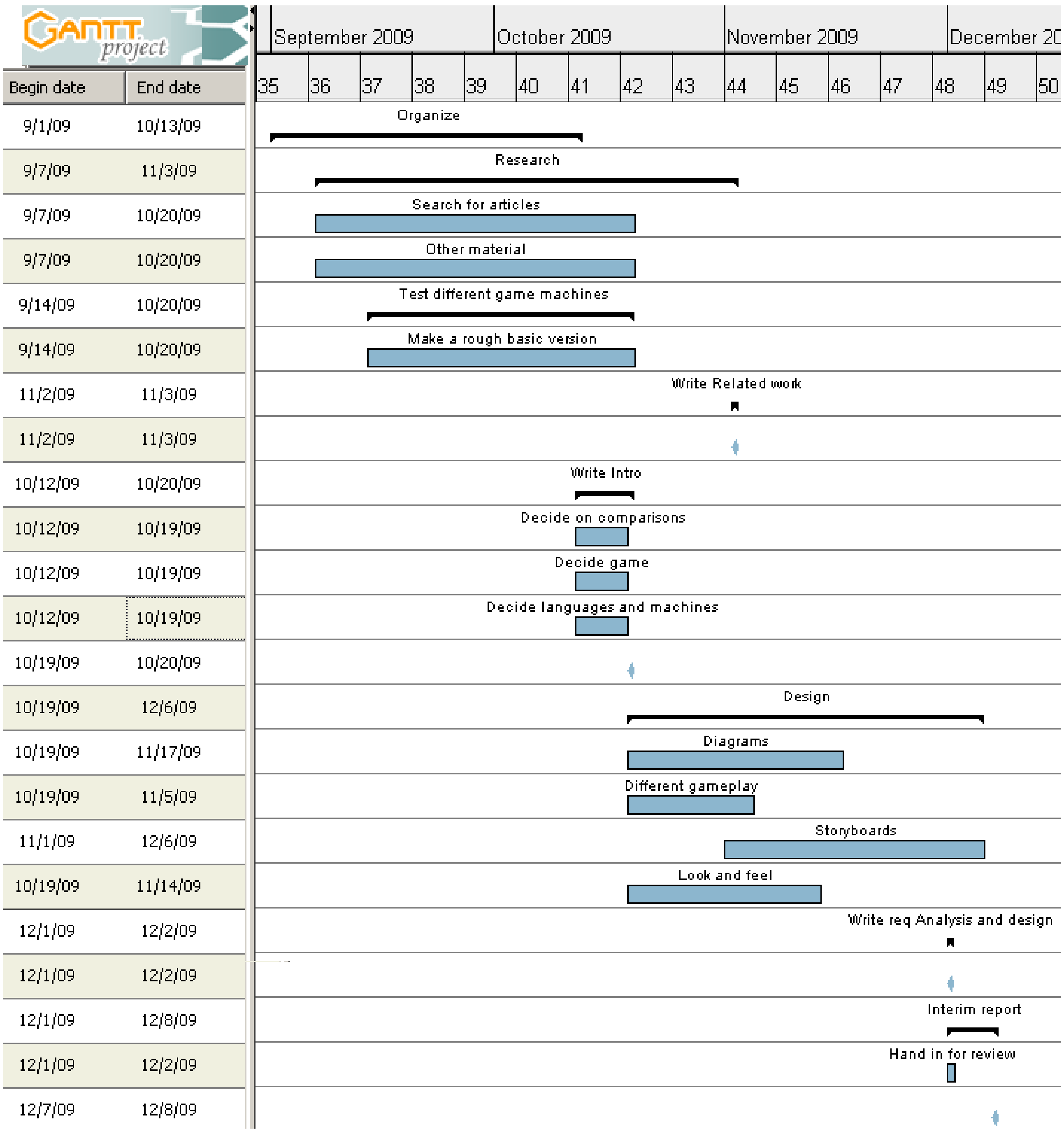
```
class Item:
    """

    Class for a piece. This just handles loading the model and setting initial
    position and color
    -- chess tutorial

    """

    def __init__(self, square, color):
        self.model = "Textures/pawn"
        self.obj = loader.loadModel(self.model)
        self.obj.reparentTo(render)
        self.obj.setColor(color)
        self.obj.setPos(SquarePos(square))
        self.color = color
        self.mugwumpNr = -1
        self.searched = False
        self.location = square

    def setColor(self, color):
        self.obj.setColor(color)
        self.color = color


# Run the game
g = Game()
run()
```

```
package code
{
	import flash.display.MovieClip;
	import flash.text.*;
	import flash.events.*;


	public class MugwumpsGraph2 extends MovieClip
	{
		import fl.controls.Label;
		import fl.controls.TextArea;

		var remaining:int = 4;
		var mugwumpNr:int = 4;
		var bushNr:int = 15;
		var tries:int = 0;
		// Create an array,
		// 2 dimensions, x and y
		var sizeW:int = 80;
		var sizeH:int = 50
		var map:Array = new Array();
		var mapX:int = 800/sizeW;
		var mapY:int = 480/sizeH;

		// location of visible objects/bushes
		var bushes:Array = new Array();
		var bushGraph:Array = new Array();

		var titleLabel:Label = new Label();
		var labelInstructions:Label = new Label();
		var textMap:Label;

		var mouseclicked:Label = new Label();

		public function MugwumpsGraph2()
		{
			trace("Mugwumps running...");

			// top text
			titleLabel.text = "Mugwumps Game";
			titleLabel.move(10,10);
			titleLabel.autoSize = TextFieldAutoSize.LEFT;
			addChild(titleLabel);

			// mouse text
			mouseclicked.text = "";
			mouseclicked.move(10,560);
			mouseclicked.autoSize = TextFieldAutoSize.LEFT;
			addChild(mouseclicked);

			// create 'empty' array
			fillMap();

			// place bushes on map
			for (var j = 0; j < bushNr; j++) {
				placeBushes(j);
			}
			// place mugwumps in bushes
			for (var i:int = 0; i < remaining; i++) {
				trace("Placing Mugwump "+i);
				placeMugwumps();

			}

			// Instruction text
			labelInstructions.text = "Guess the coordinates of the nearest mugwump";
			labelInstructions.move(210,480);
			labelInstructions.autoSize = TextFieldAutoSize.LEFT;
			addChild(labelInstructions);

			// Text for map
			textMap = new Label();
			textMap.move(650,450);
			textMap.autoSize = TextFieldAutoSize.LEFT;
			printMap();
			addChild(textMap);

			stage.addEventListener(MouseEvent.CLICK, mouseClicked);
		}

		// fill the map array with arrays to do x,y coordinates
		// then fill all with 0
		public function fillMap():void
		{
			for (var i:int = 0; i < mapX; i++) {
				trace("Map X"+i)
				map[i] = new Array();
				for (var j:int = 0; j < mapY; j++) {
					trace("Map Y"+j)
					map[i].push(0);
				}
			}
		}

		// Place visible objects/bushes
		public function placeBushes(j) {
			// find a random location
			var posX:int = randomNumber(0,mapX-1);
			var posY:int = randomNumber(0,mapY-1);

			if (map[posX][posY] == 0) {
				// draw a bush using a label
				// TODO: make this a new graphic image of a bush
				bushes[j] = new Label();
				bushes[j].move(posX*sizeW, posY*sizeH);
				bushes[j].text =
"BUSHBUSHBUS\nBUSHBUSHBUS\nBUSHBUSHBUS\nBUSHBUSHBUS";
				bushes[j].autoSize = TextFieldAutoSize.LEFT;
				addChild(bushes[j]);
				map[posX][posY] = 1;
				bushGraph[j] = new Graph(posX*sizeW, posY*sizeH);

				//addChild(bushGraph[j]);

			}
		else
			placeBushes(j);
	}

	public function placeMugwumps():void
	{
		var posX:int = randomNumber(0,mapX-1);
		var posY:int = randomNumber(0,mapY-1);

		if (map[posX][posY] == 1) {
			// if empty, hide mugwump
			map[posX][posY] = 2;
			//bushes[j].text = "MUGWUMP";
		}
		else
			placeMugwumps();
	}
	public function printMap():void
	{
		trace("Print Map...");
		var val:int = 0;
		// print the array
		textMap.text = "  ";
		for (var i:int = 0; i < mapX; i++) {
			textMap.text = textMap.text + " | " + i;
		}
		textMap.text = textMap.text + "\n";
		for (var yi:int = 0; yi < mapY; yi++) {
			textMap.text = textMap.text + yi;
			for (var xi:int = 0; xi < mapX; xi++) {
				val = map[xi][yi];
				if ((val <= 2))
				{
					textMap.text = textMap.text + " |  ";
				}
				else if (val == 3)
				{
					textMap.text = textMap.text + " | o";
				}
				else if (val == 4)
				{
					textMap.text = textMap.text + " | x";
				}
				else
				// something screwy going on
					textMap.text = textMap.text + " | ?";

			}
			textMap.text = textMap.text + "\n";
		}
	}

	public function mouseClicked(event:MouseEvent):void {
		var posY:int = Math.round(event.stageY)/80;
		var posX:int = Math.round(event.stageX)/80;

		if ((posX >= mapX) || (posY >= mapY))
		{
				labelInstructions.text = "You need to click on the map to guess";
				return;
			}

			labelInstructions.text = "You guessed " + posX + " x " + posY +".";

			if (map[posX][posY] == 2)
			{
				labelInstructions.text = labelInstructions.text + "\nYou found a Mugwump.";
				map[posX][posY] = 4
				remaining--;
				tries++;
			}
			else if (map[posX][posY] == 1)
			{
				labelInstructions.text = labelInstructions.text + "\nThere was no Mugwump
there,\ntry searching another bush.";
				map[posX][posY] = 3;
				tries++;
			}
			else if (map[posX][posY] < 1)
			{
				labelInstructions.text = labelInstructions.text + "\nThere was no Mugwump
there,\ntry looking in a bush.";
				map[posX][posY] = 3;
				tries++;
			}
			else
			{
				labelInstructions.text = labelInstructions.text + "\nYou have already guessed there";
			}
			if (remaining > 0)
			{
				checkNearest(posX,posY);
			}
			else
				labelInstructions.text = labelInstructions.text + "\nYou have found all mugwumps.";
			mouseclicked.text = "Tries: " + tries + "\nFound: " + (mugwumpNr - remaining);
			printMap();
		}

		public function checkNearest(posX, posY):void
		{

			var nearest:int = 1000
			var distX:int = 0;
			var distY:int = 0;
			var newNear:int = 0;
			var northSouth:String = "";
			var westEast:String = "";

			for (var xi:int = 0; xi < map.length; xi++) {
				for (var yi:int = 0; yi < map[xi].length; yi++) {
					if ((map[xi][yi] == 2) && ((xi != posX) || (yi != posY)))
					{
						distX = xi - posX;
						trace(distX);
						distY = yi - posY;
```

```
                                                trace(distY);
                                                newNear =  Math.sqrt(distX*distX + distY*distY);
                                                if (newNear < nearest)
                                                {
                                                        nearest = newNear;
                                                        if (distX < 0)
                                                        {
                                                                westEast = "West";
                                                        }
                                                        else if (distX > 0)
                                                        {
                                                                westEast = "East";
                                                        }
                                                        else
                                                                westEast = "";
                                                        if (distY < 0)
                                                        {
                                                                northSouth = "North";
                                                        }
                                                        else if (distY > 0)
                                                        {
                                                                northSouth = "South";
                                                        }
                                                        else
                                                                northSouth = "";

                                                }
                                        }
                                }
                        }
                        labelInstructions.text = labelInstructions.text + "\nNearest is " + nearest + " squares away to
the "
                                + northSouth + westEast +".";
                }

                /**
                * Generate a random number
                * @return Random Number
                * @error throws Error if low or high is not provided
                */
                function randomNumber(low:Number=NaN, high:Number=NaN):Number
                {
                        var low:Number = low;
                        var high:Number = high;

                        if(isNaN(low))
                        {
                        throw new Error("low must be defined");
                        }
                        if(isNaN(high))
                        {
                        throw new Error("high must be defined");
                        }

                        return Math.round(Math.random() * (high - low)) + low;
                }
        }
}
```

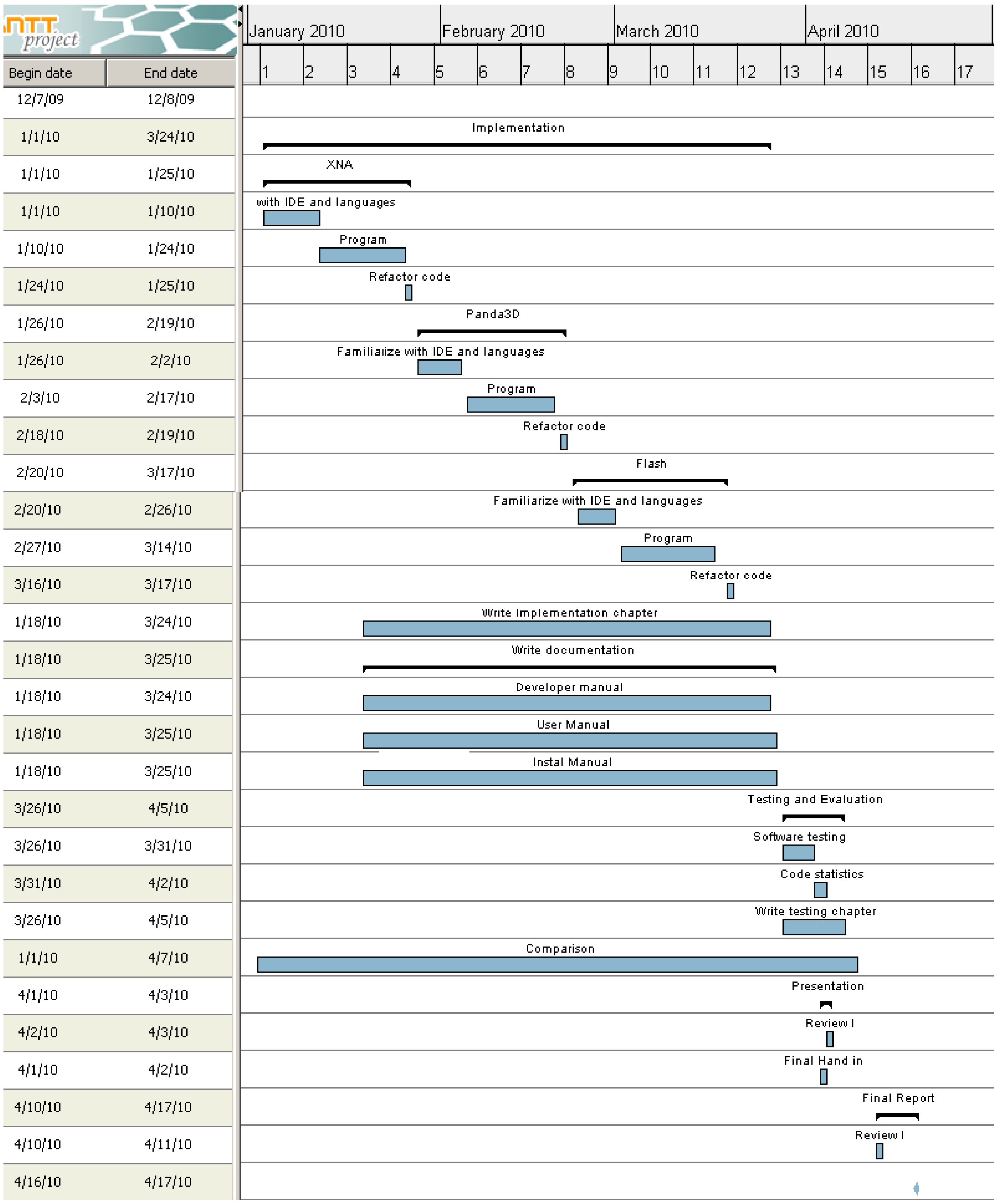
Appendix A - Project plan

First parts of the project, research and design, are completed. What follows is implementation, testing and comparison. Implementation will have very strict time limits because there are three different implementations to do. Testing will be cut shorter than would be ideal if the software were to be released for use to give more time for implementaiton and comparison of engines.



Begin date	End date	
9/1/09	10/13/09	Organize
9/7/09	11/3/09	Research
9/7/09	10/20/09	Search for articles
9/7/09	10/20/09	Other material
9/14/09	10/20/09	Test different game machines
9/14/09	10/20/09	Make a rough basic version
11/2/09	11/3/09	Write Related work
11/2/09	11/3/09	
10/12/09	10/20/09	Write Intro
10/12/09	10/19/09	Decide on comparisons
10/12/09	10/19/09	Decide game
10/12/09	10/19/09	Decide languages and machines
10/19/09	10/20/09	
10/19/09	12/6/09	Design
10/19/09	11/17/09	Diagrams
10/19/09	11/5/09	Different gameplay
11/1/09	12/6/09	Storyboards
10/19/09	11/14/09	Look and feel
12/1/09	12/2/09	Write req Analysis and design
12/1/09	12/2/09	
12/1/09	12/8/09	Interim report
12/1/09	12/2/09	Hand in for review
12/7/09	12/8/09	

Illustration 1: Timeline up to end of Autumn semester



Begin date	End date	
12/7/09	12/8/09	
1/1/10	3/24/10	Implementation
1/1/10	1/25/10	XNA
1/1/10	1/10/10	with IDE and languages
1/10/10	1/24/10	Program
1/24/10	1/25/10	Refactor code
1/26/10	2/19/10	Panda3D
1/26/10	2/2/10	Familiarize with IDE and languages
2/3/10	2/17/10	Program
2/18/10	2/19/10	Refactor code
2/20/10	3/17/10	Flash
2/20/10	2/26/10	Familiarize with IDE and languages
2/27/10	3/14/10	Program
3/16/10	3/17/10	Refactor code
1/18/10	3/24/10	Write implementation chapter
1/18/10	3/25/10	Write documentation
1/18/10	3/24/10	Developer manual
1/18/10	3/25/10	User Manual
1/18/10	3/25/10	Instal Manual
3/26/10	4/5/10	Testing and Evaluation
3/26/10	3/31/10	Software testing
3/31/10	4/2/10	Code statistics
3/26/10	4/5/10	Write testing chapter
1/1/10	4/7/10	Comparison
4/1/10	4/3/10	Presentation
4/2/10	4/3/10	Review I
4/1/10	4/2/10	Final Hand in
4/10/10	4/17/10	Final Report
4/10/10	4/11/10	Review I
4/16/10	4/17/10	

Illustration 2: Timeline up to end of Spring semester

Appendix B – Project Suggestion

Title: Comparison of game engines and languages

Supervisor: Dr Andy Brooks

Development: XNA Game Studio, PANDA3D, Scala language, and possible others.

Description: The aim of this project is to re-implement one or more classic computer games (the Java versions) using one or more game engines or emerging Web 2.0 languages. Comparisons will be made of the facilities offered by the development environments and the code produced using a variety of metrics. The facilities offered by the game engines and languages will be exploited to create a richer game experience e.g. through the use of graphics or an NPC that the game player can interact with.

References:

„ACEY-DEUCEY is a fun card game. EVEN WINS is one of those 'remove the markers' games. MUGWUMP asks you to find the hidden monsters in a grid. LUNAR LANDER lets you land safely on the moon. FROWN is a fun dice game you play against the computer. And, JOT is an addictive word guessing game.“

http://www.kidwaresoftware.com/javagames.htm

http://creators.xna.com/en-US/downloads (XNA)

http://catpages.nwmissouri.edu/m/ferg/VideoGameDevelopment.htm (XNA)

http://www.xnatutorial.com/ (XNA)

http://xnaessentials.com/Default.aspx (XNA)

http://www.panda3d.org/features.php (PANDA3D)

http://www.scala-lang.org/ (Scala)

http://www.computerworld.com.au/company/13389182/scala/articles (Scala)

http://www.plasmaworks.com/slag (Slag)

Appendix D – Brainstorming

I want to create a game I and my sisters would have enjoyed when we were kids, and that we could still enjoy as grown women. I don't want to just decide out of the blue but to find reasons why, from those in industry or education. Mainly aimed as enjoyement for kids, where they might learn something without it being stuffed down their throat.

I played the education games when I was a kid at school, I probably learned some math and spelling from them but I did not like them. I felt that the grown-ups were forcing me to imagine something as play when it wasn't. I had more fun playing 'school' with my sisters where we used old schoolbooks to take turn to teach. Mostly because we did it because we wanted to when we wanted to, not because it was now math class and the math teacher was somehow trying to trick us into 'playing with math'.

Because of the idea of creating a game that I would have enjoyed as a kid and would enjoy now, the main target audience will be kids and keep in mind to not dumb it down.

The Mugwumps are the 'chosen ones'. Start with a creature (Muggy) finding the player (known as 'PC', choose between girl or boy at start of game) and telling PC that the Mugwumps need to be found. Gives a magic stone that shows in what dirrection the Mugwumps are.

Each 'level' is a puzzle of some sort. First few are just to click different areas so that the sprite moves there and sees if there might be mugwump there. Muggy is with the player to help and give hints. Simulates the hit and miss of the original game. Other can be: showing the mugwumps through a maze, remove sound/sight/stone/muggy to make it difficult, timer, find items, help build town/home/something???.

Features I want:

- puzzles
  - Find mugwump
  - guide mugwumps
  - find the password?
  -
  - Use item to get through?
- building
- collecting
- customizing
- cute
- slow action

Take stuff with from room? Find stuff instead of mugwumps? With mugwumps? Levels where stuff is found? Buy stuff?

sound:

- use 'babble' instead of real words – no reliance on language

- lots of squee and uhoh – think teletubbies

text:

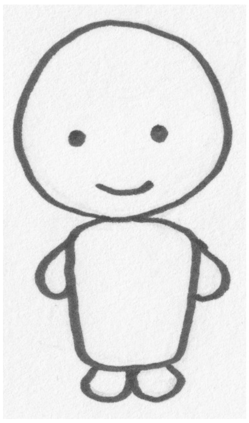
- short description of what to do “oh no help mugwumps reach pink thing”
- have pictures in text? With text? [picture of mugwump] [arrow] [pink thing]
- large letters
- simple
- language file

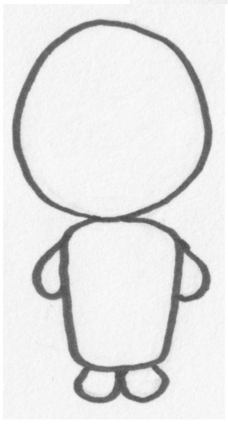
pictures

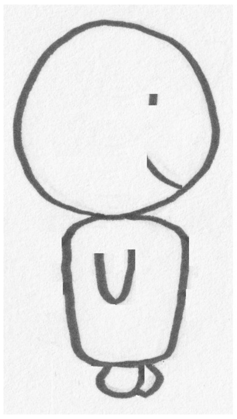
- “large” - simple - cartoony
- colourful
- can color own?
- Not manga

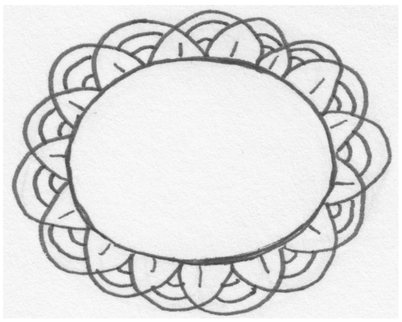
Features I don't want

- obvious teaching elements such as: 'oh noes what is 2+2' or 'spell this to get past'
- too much text / talk









A Comparison of Game Engines and Languages

Appendix C

Software Requirement Specification

Rósa Dögg Jónsdóttir
University of Akureyri
December 2009

Table of Contents

1 INTRODUCTION

This specification establishes the functional, performance, and development requirements for a game called Mugwumps. The game will be created in three different game engines: Microsoft's XNA, Panda3D and Macromedia's Flash.

1.1 PURPOSE

Developing the game is part of a final year project 2009, created by Rósa Dögg Jónsdóttir and supervised by dr. Andy Brooks.

1.2 THE GAME

The Mugwumps game is an interactive computer game with a graphic interface. In the game the user solves a series of hide and seek puzzles, and collects rewards. The puzzles are of different difficulty and can be replayed.

1.3 INTENDED AUDIENCE

The game is for young children, ages 6-8. The user does not need any special knowledge or skill aside from mouse control and basic reading skills. The game can be played by older users.

1.4 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

Mugwump – a small cartoony creature.
The player – the user of the game.
PC – player character.
NPC – non-player character.
The game – a computer game named Mugwumps.
Magic stone – a circular object in the game that gives hints to where the hidden objects are. Objects are represented by colour. The colour grows stronger the closer the PC is to the object.
Muggy – a mugwump NPC that follows the PC through the game and gives hints.
Score – how often the player had to guess where the object was hiding.

1.5 DOCUMENT OVERVIEW

Section 1 describes the purpose of the game and this document. It lists the definitions, acronyms and reference documents. Section 2 descripes the game overall characteristics. Section 3 lists specific requirements including use cases. This document is an appendix to the final year project report.

2 OVERALL DESCRIPTION

2.1 PERSPECTIVE

Back when computer games were text based, a game called Mugwumps was created in BASIC programming language. In it the player would guess the location of hidden Mugwumps based on hints on how far away they were. This project takes that game idea and transfers it into a graphical game with 3 puzzles of varius difficulties. Muggy, a mugwump non-player character (NPC), will follow the player character (PC) around to provide hints on what the PC should do. The PC will have a visual hint from a magic stone.

2.2 GAME FUNCTIONS

The player will create a PC, customizing the name. Muggy will join the PC at the start of the game and stay with him to the end. Muggy will give the PC a magic stone at the start of the game. The game will have three chapters, each one has a hide and seek puzzle for the PC to solve. Player will use a mouse click to guess the location of hidden objects. When all chapters have been completed the player can play any chapter again. The game keeps track of the best score for each chapter.

2.3 CONSTRAINTS

There is limited time and resources to create the game. The game must be open for future development, where the type of puzzles and rewards can be extended or replaced to give different game variations.

2.4 ASSUMPTIONS AND DEPENDENCIES

Different game engine versions of the game will require different software to run (see 3.1.3 Software Interfaces). It is assumed that the creator and supervisor will have access to that software.

3 OVERVIEW

3.1 EXTERNAL INTERFACE REQUIREMENTS

3.1.1 User Interfaces

The user recieves information from a screen, both graphic and text. The user uses a mouse to point and click to give commands. The user will require the use of a keyboard at the start of the game to type in his name.

3.1.2 Hardware Interfaces

Keyboard, mouse and screen. Soundcard is optional.

3.1.3 Software Interfaces

Each version of the game will require different software.

3.1.3.1 XNA

Requires Microsoft windows XP, Vista or 7 and .NET framework 2.0 or higher

3.1.3.2 Panda3D

Should run on any Unix or Windows based machine. Requires either directX or openGL.

3.1.3.3 Flash

Runs through a web browser. Requires flash player.

3.2 FUNCTIONAL REQUIREMENTS

The game will have different menus or game windows during different stages of the game. The game will always listen for mouse input from the player. The game will at certain stages listen to keyboard input from the player.

3.2.1 Game menu

The game menu is displayed when the game is loaded and if a chapter is closed. It will provide options to create a new game (with new name), load a game, display help or quit.

3.2.2 Chapter menu

When a game is created or loaded the player will see a chapter menu. If a chapter has been completed it can be selected as well as the next chapter in line. If no chapters have been completed only chapter 1 is selectable. It provides option to quit to game menu and display help.

3.2.3 All chapters

In each chapter the game will hide one or more objects (defined in each chapter) in random locations (defined in each chapter). In each chapter the PC will use the mouse to click on a space on the screen to guess the hidden object's location. In each chapter Muggy will provide hints of what the PC must do. In each chapter the magic stone will display colours to give hints to the location of objects. When the PC has found all hidden objects in a chapter, the PC can either play the same chapter again or go to the next chapter. All chapter windows display a help and quit (to chapter menu) options.

3.2.4 Chapter 1

The first chapter is for practice. The scenery is a field. The player's goal is to find Muggy, using the magic stone and hints that Muggy shouts out from his hiding place. Muggy will be hidden randomly behind one of 4 different objects (a tent, a tree, a bush and a stone). The objects are arranged randomly on a grid that covers the whole field. When the player guesses the location and is wrong the object changes to show it has been searched.

3.2.5 Chapter 2

The second chapter is intermediate difficulty. The scenery is a tent camp on a field. The player's goal is to find three different Mugwumps that are all in hiding at the same time. The Mugwumps are represented with three different colours in the magic stone. The Mugwumps are hidden randomly behind one of 12 different objects (a well, a wagon, tents, boxes and barrels). The objects are arranged randomly on a grid that covers the whole field. When the player guesses the location and is wrong the object changes to show it has been searched. When each Mugwump is found he will stay visible at the place where he was hiding. When all Mugwumps have been found the player recieves a reward.

3.2.6 Chapter 3

The third chapter is the most difficult. The scenery is a field. The player's goal is to find a hidden treasurechest. The treasurechest is hidden in a random location on a grid that covers all of the field. There are no visual hints to the hidden location except the magic stone and Muggy's text. When the player guesses a location a hole appears, as if he had dug one. When the player has found the treasurechest he recieves a reward.

3.2.7 Muggy

Muggy the Mugwump gives the player hints and urges him onward in the game. At the start of each chapter he tells the rules. After every 30 seconds of idle time from the player, Muggy displays a random hint. For each chapter Muggy has a selection of hints, each hint has a frequency number. Higher frequency number means that the hint is more likely to appear in the random selection. When a hint has been shown the frequency number goes down. If the player clicks an object that has been searched, Muggy warns that it has been and he should look elsewhere. Muggy translates the information the magic stone shows into text.
Hints are of type:

- Which way the object is hiding.
- How close it is (next to it, very close, close, a bit far, far)
- Tell to click objects on the screen.
- Tell what is hiding.
- Remind what has been searched

3.3 PROGRAMMING SPECIFICATIONS

The game will be programmed in three different languages: C# (Microsoft's XNA engine), Python (Panda3D) and ActionScript (Flash). There will be some difference between each programming languages in class and method hierarchy. There are certain required functions for the game regardless of programming language.

- Graphic management
- Menu management
- Game loop
  - Manage visible and hidden objects
  - Manage magic stone and Muggy hints
  - Track player activities.
- Player information management

Use of these functions can be represented with use cases.

3.3.1 Use Cases

3.3.1.1 Start game software (use case 1)

1 User: Selects to run software
  1.1 XNA: Run .exe
  1.2 Panda3D: Run start script
  1.3 Flash: Open web browser on game page.
2 Computer: Checks for components required to run.
3 Computer: Checks previous player information.
4 Computer: Load in menu graphics
5 Computer: Load in player information (if it exists)
6 If any action fails
  6.1 Computer: Display error
  6.2 Computer: Close software
  6.3 User: Try again or Quit.
7 If all actitons succeed
  7.1 Computer: Display game menu (use case 2).

3.3.1.2 Game menu (use case 2)

1 Computer: Display text options
  1.1 New : create a new game
    1.1.1 Use case 5
  1.2 Load : load a game
    1.2.1 Use case 6
  1.3 Help : go to help menu
    1.3.1 Use case 3
  1.4 Quit : close the game
    1.4.1 Use case 4
2 Computer: Listen for Mouseclick.
3 While player is idle
  3.1 Computer: Wait
4 Player: Select option using mouse
  4.1 <Go to approrpiate use case>
5 If any action fails

```
    5.1 Computer: Display error
    5.2 Computer: Return to game menu
        5.2.1 If action fails
            5.2.1.1 Computer: Display error
            5.2.1.2 Computer: Close software
    5.3 Player: Try again or Quit
```

3.3.1.3 Help Menu (use case 3)

```
1 Computer: Lists help topics.
2 Computer: Displays quit option.
3 Computer: Listen for mouseclick
4 While player is idle
    4.1 Computer: Wait
5 Player: Select help topic
    5.1 Computer: Display text for help topic chosen
6 Player: Select quit
    6.1 Computer: Return to previous screen
7 If action fails
    7.1 Computer: Display error
    7.2 Computer: Quit to previous screen
        7.2.1 If action fails
            7.2.1.1 Computer: Display error
            7.2.1.2 Computer: Close software
```

3.3.1.4 Quit (use case 4)

```
1 Computer: Display confirmation box
2 While player is idle
    2.1 Computer: Wait
3 Player: Select cancel
    3.1 Computer: go back to previous screen
4 Player: Select quit
    4.1 If new player information was generated
        4.1.1 Computer: save player information
    4.2 Computer: Quit to previous menu
5 If action fails
    5.1 Computer: Display error
    5.2 Computer: Close software
```

3.3.1.5 New Game (use case 5)

```
1 Computer: Display text box for name
2 Computer: Display help and cancel
3 Computer: Listen to mouseclick and keyboardstroke
4 While player is idle
    4.1 Computer: Wait
5 Player: Type in name
6 Player: Select continue
    6.1 Computer: Create new player information identified by name.
    6.2 Computer: Chapter menu (use case 7)
7 Player: Select cancel
```

```
    7.1       Computer: return to game menu
8   If player selects continue without typing in name
    8.1       Computer: Display warning
    8.2       Computer: Return to New Game
9   Player: Select help
    9.1       Use case 3
10        If action fails
    10.1      Computer: Display error
    10.2      Computer: Return to game menu
        10.2.1    If action fails
            10.2.1.1  Computer: Display error
            10.2.1.2  Computer: Close software
```

3.3.1.6 Load Game (use case 6)

```
1   Computer: Display available game instances to load
    1.1       Based on player information
2   Computer: Display quit and help
3   Computer: Listen to mouseclick
4   While player is idle
    4.1       Computer: Wait
5   Player: Select game instance
    5.1       Computer: Display Chapter menu (use case 7)
6   Player: Select help
    6.1       Use case 3
7   Player: Select quit
    7.1       Use case 4
8   If action fails
    8.1       Computer: Display error
    8.2       Computer: Return to game menu
        8.2.1     If action fails
            8.2.1.1   Computer: Display error
            8.2.1.2   Computer: Close software
```

3.3.1.7 Chapter menu (use case 7)

```
1   Computer: Display available chapters and top scores
    1.1       Display chapter 1 always
    1.2       Display chapter 2 if chapter 1 is completed
    1.3       Display chapter 3 if chapter 2 is completed
2   Computer: Display quit and help
3   Computer: Listen for mouseclick
4   While player is idle
    4.1       Computer: Wait
5   Player: Select chapter
    5.1       Computer: Load chapter (use case 8)
6   Player: Select help
    6.1       Use case 3
7   Player: Select quit
    7.1       Use case 4
8   If action fails
```

```
    8.1       Computer: Display error
    8.2       Computer: Return to load menu
        8.2.1     If action fails
            8.2.1.1   Computer: Display error
            8.2.1.2   Computer: Close software
```

3.3.1.8 Load chapter (use case 8)

```
1 Computer: Check for components required to run
2 Computer: Determines location of visible objects
3 Computer: Determines location of hidden object(s)
4 Computer: Create new score instance
5 Computer: Display graphics
    5.1       board, visible objects, magic stone, Muggy's text area, player sprite, help icon,
        quit icon
6 Muggy (Computer): Descripe chapter objectives
7 Computer: Start game loop (use case 9)
9 If action fails
    9.1       Computer: Display error
    9.2       Computer: Return to chapter menu
        9.2.1     If action fails
            9.2.1.1   Computer: Display error
            9.2.1.2   Computer: Close software
```

3.3.1.9 Game loop (use case 9)

```
1 Computer: Update graphics if required
2 Magic stone (Computer): Display dirrection and distance of hidden object(s).
3 While player is idle
    3.1       Computer: Wait
    3.2       Muggy (Computer): Every 30 seconds display new hint
4 Player: Select visible object on board
    4.1       Computer: Move player sprite to object
    4.2       Computer: Check if object has been searched before
        4.2.1     If yes: Muggy (Computer) reminds that the object has been searched.
        4.2.2     If no: Continue
    4.3       Computer: Check if object has a hidden object
        4.3.1     If yes:
            4.3.1.1   Muggy (Computer): Congratulates player
            4.3.1.2   Computer: Changes status of visible object to 'searched'
            4.3.1.3   Computer: Changes status of hidden object to 'found'
            4.3.1.4   If this was the last object to be found:
                4.3.1.4.1  Computer: in 8 seconds end chapter (use case 10)
            4.3.1.5   If this was not the last object to be found:
                4.3.1.5.1  Muggy (Computer): in 8 seconds displays new hint
        4.3.2     If no:
            4.3.2.1   Muggy (Computer): Is sad but encourages player to keep looking.
            4.3.2.2   Computer: Changes status of visible object to 'searched'
5 Player: Select empty area on board
    5.1       Muggy (Computer): Displays hint about clicking objects on the board
6 Player: Select magic stone
```

```
    6.1     Magic stone (Computer): Colours whirl
    6.2     Magic stone (Computer): Display dirrection and distance of hidden object(s)
 7  Player: Select Muggy
    7.1     Muggy (Computer): Giggles
 8  Player: Select other
    8.1     Nothing happens
 9  Player: Select help
    9.1     Use case 3
10  Player: Select quit
    10.1    Use case 4
11  If action fails
    11.1    Computer: Display error
    11.2    Computer: Return to play
        11.2.1      If action fails
            11.2.1.1    Computer: Display error
            11.2.1.2    Computer: Return to chapter menu
                11.2.1.2.1      If action fails
                    11.2.1.2.1.1    Computer: Display error
                    11.2.1.2.1.2    Computer: Close software
```

3.3.1.10 End chapter (use case 10)

1. Muggy (Computer): Display congratulations
2. Computer: Display 'play again'.
3. If the chapter that was ending is not chapter 3
   1. Computer: Display 'next chapter'.
4. Computer: Display 'quit' and 'help'
5. While player is idle
   1. Computer: Wait
6. Player: Select 'play again'
   1. Computer: Save player information
   2. Use case 8, with same chapter
7. Player: Select 'next chapter'
   1. Computer: Save player information
   2. Use case 8, with next chapter
8. Player: Select 'quit'
   1. Computer: Save player information
   2. Use case 4
9. Player: Select 'help'
   1. Use case 3
10. If action fails
    1. Computer: Save player information
    2. Computer: Display error
    3. Computer: Return to chapter menu
       1. If action fails
          1. Computer: Display error
          2. Computer: Close software

3.4 PERFORMANCE REQUIREMENTS

The game needs to run smoothly, without visible hesitation in frame rate or logical processing. Game should take no longer than 2 seconds to respond to player input.

3.5 DESIGN CONSTRAINTS

The game needs to be appealing for young children. This means bright, basic colours, simple forms, cheerful sounds and simple use of language. Colours need to be chosen to take into account colourblindness. All text presented in the game for the user to see must be easy to edit to change languages.

3.5.1 Standards Compliance

Any code produced by the creator must follow basic standards of readability according to each programming language.

3.6 SOFTWARE SYSTEM ATTRIBUTES

3.6.1 Reliability

The game should be playable from start to finish without errors. Score should be kept between instances of the game.

3.6.2 Availability

The game will be available with the final year project report.

3.6.3 Security and Privacy

No special security or privacy will be applied.

3.6.4 Maintainability

Code and any documentation will be available with the final year project report so that the game can be expanded or changed.

Appendix F - Use Cases Reworked

For XNA and Panda3D

1 XNA

1.1 Instal Game

1.1.1 Run setup.exe

1.1.2 When Application Warning pops up, select 'Install'.

1.1.3 Game starts automatically.

1.2 Start Game

1.2.1 Navigate to Start menu – All Programs – MugwumpsGameXNA

1.2.2 Click (run) MugwumpsGameXNA

1.2.3 Main Menu displayed. Use arrow keys to select options, use enter key to activate.

1.2.3.1 Play game

1.2.3.1.1 Start a new game instance

1.2.3.2 Options

1.2.3.2.1 Fake options screen

1.2.3.3 Exit

1.2.3.3.1 Closes the game

1.3 Play game

1.3.1 Muggy displays starting hints. Visible objects are bushes, computer has hidden Mugwumps randomly in visible objects. Player sprite is besides Muggy's hint.

1.3.2 User can press ESC at any time to view Pause Menu

1.3.3 User can use mouse click at any time to:

1.3.3.1 Select visible object

1.3.3.1.1 If the Player Sprite is on it's way to a previous location, Muggy warns that they are on the way and do nothing.

1.3.3.1.2 Else, if the Player Sprite was not moving, start moving the Player Sprite to the location and Muggy says that they will search there.

1.3.3.1.2.1 When Player Sprite is on location, Muggy will announce hint.

1.3.3.2 Select an empty area of the map

1.3.3.2.1 If the Player Sprite is on it's way to a previous location, Muggy warns that they are on the way and do nothing.

1.3.3.2.2 Else, if the Player Sprite was not moving, start moving the Player Sprite to the location and Muggy says that they will search there.

1.3.3.2.2.1 When Player Sprite is on location, Muggy will announce hint.

1.3.3.3 Select outside of map

1.3.3.3.1 Muggy reminds the user to select something on the map.

1.4 Options

1.4.1 A fake option menu. User can use keyboard arrows to select objects or press ESC to go back.

1.5 Pause Menu

1.5.1 Resume game

1.5.2 Quit game

1.5.2.1 Displays confirmation dialog. Press enter to quit and return to Main Menu. Press ESC to return to Pause Menu

1.5.3 New game

1.5.3.1 Loads a new instance of the game, destroying the other.

2 Panda3D

2.1 Instal Game

2.1.1 Make sure python is installed on the computer.

2.1.2 Copy MugwumpsGraphic folder to c:\

2.1.2.1 Note: Can place at another location but then start.bat needs to be edited.

2.2 Start Game

2.2.1 Run start.bat located in the MugwumpsGraphic folder.

2.2.2 Game starts.

2.2.2.1 Note, a command window opens in the background. It is safe to ignore it.

2.3 Play game

2.3.1 Starting hint is displayed. Visible objects are pawns, computer has hidden Mugwumps randomly in visible objects.

2.3.2 User can press ESC at any time to quit

2.3.3 User can use mouse click at any time to:

2.3.3.1 Select visible object

2.3.3.1.1 If there is a Mugwump hiding in it the game will announce that and display a hint of the nearest Mugwump.

2.3.3.1.2 If there is not a Mugwumpdisplay location of nearest Mugwump hint.

2.3.3.2 Select an empty area of the map

2.3.3.2.1.1 Game displays location of nearest Mugwump hint.

2.3.3.3 Select outside of map

2.3.3.3.1 Muggy reminds the user to select something on the map.

2.3.4 When all Mugwumps have been found it is announced.