Table of contents :

1 Introduction 3
2 SAGA 3
   2.1 The SAGA project 3
   2.2 Our project 3
3 Organization plan 3
   3.1 Working Arrangements 4
   3.2 Methodology 4
      3.2.1 Roles 4
      3.2.2 Sprints 4
   3.3 Work hours 4
   3.4 Expected deliverables 5
   3.5 Tools 6
4 Work plan 6
   4.1 Introduction 6
   4.2 Sprint plan 6
5 Risk Analysis 7
   5.1 Occurrences 8
6 Design 8
   6.1 GOLOG planner 8
   6.2. Unity3D Viking Village 9
      6.2.1 Initial State 9
      6.2.2. Our goals for viking village 9
7 Project backlog 9
8 Progress report 11
   8.1 Preparation 11
   8.2 Sprint 1 11
      8.2.1 Backlog 11
      8.2.3 Burndown 11
      8.2.4 Retrospectives 12
   8.3 Sprint 2 12
      8.3.1 Backlog 12
      8.3.3 Burndown 13
      8.3.4 Retrospectives 13
   8.4 Sprint 3 13
      8.4.1 Backlog 13
      8.4.3 Burndown 14
1 Introduction
In this report we go over the progress and details of the SAGA project. This project is a final project in University of Reykjavik and the SAGA project is owned by CADIA the Artificial Intelligence research center at University of Reykjavik. We will also outline the organization and work plan for our work in the SAGA project.

2 SAGA
Compared to other kinds of entertainment, computer games are a unique medium in that they offer players a wide variety of opportunities to influence what happens next. Meanwhile, stories are usually pre-defined: no matter what the audience does, the same sequence of events occurs. When stories are put into games, however, players can influence the game in ways that disrupt a pre-defined story, resulting in a story that no longer makes sense.

2.1 The SAGA project
The SAGA project combines different techniques in Artificial Intelligence (AI) to manage this problem by automatically simulating and adapting stories while the game is underway. Since Fall 2014, many students have been implementing the SAGA project as a software system; this system can currently generate stories made up of individual characters’ actions and display them in a simple Unity3D environment. The SAGA project uses a planner among other techniques to generate stories.

2.2 Our project
Recently the planner was replaced by a GOLOG planner that is capable of generating more varied and higher quality stories. However, the GOLOG planner is slow due to the naive form of search that it performs, and it has no way to differentiate between the quality of the stories that it generates. Furthermore, the current 3D Unity environment lacks sufficient content and capabilities to properly demonstrate the benefits of the SAGA system. We seek to extend the system by 1) improving the quality of the stories generated and improving the efficiency of the planner, and 2) improving the Unity3D environment, called Viking Village, to make it more interesting for the user and to better present the SAGA system.

3 Organization plan
This chapter will outline how we plan on working on this final project which includes working arrangement, methodologies used and when we schedule to meet together to work on the project.
3.1 Working Arrangements

We have a workspace in Cadia, the artificial intelligence facility of Reykjavík University. There each of us has a desk where we sit together. We are located on the same floor as the other people who have worked on the SAGA system so we do not have to go far if we have any questions. We will also have access to any of the meeting rooms in the facility.

3.2 Methodology

The team will use the Scrum methodology to manage the project in an agile manner.

3.2.1 Roles

The roles for this project are these two:

- Product Owner: David Thue
- Scrum Master: Kári Eiríksson

Cadia is the owner of this project and David Thue is our contact in Cadia.

3.2.2 Sprints

The team will be running two week long sprints until, when nearing the 3 week period the sprints will be shortened to 1 week. A sprint will start on a Friday and end on a Thursday two weeks after. The following Friday the team will meet with David Thue and Stephan Schiffel for a scrum review and the team will meet up for a retrospective and a scrum planning meeting.

<table>
<thead>
<tr>
<th>Sprint 1</th>
<th>Start date</th>
<th>End date</th>
<th>Sprint 6</th>
<th>Start date</th>
<th>End date</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.March</td>
<td>24.March</td>
<td>6.April</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.3 Work hours

We have dedicated 16 hours per week to work on the project together during the 12 week program:

- Thursdays from 12 to 16
- Fridays from 12 to 16
- Weekends, at least 8 hours.
During the 3 week program we plan to spend at least 8 hours each day working on the project. However individual work may be done in between the sessions where we meet together.

3.4 Expected deliverables

First hand in (25.January):
- Work schedule/organization plan
- Initial draft for work plan

For the first status meeting (12.February):
- Progress report
- Risk analysis
- Initial draft for design document
- Work plan

For second status meeting (11.March):
- Progress report
- Work plan
- Showcase a planner implementation

Final status meeting (3.May):
- Progress report
- Research paper for golog planner and heuristic

Project deadline (12.May):
- Final report
- Research paper
- Code
3.5 Tools

We’re using git for version control. For programming we use eclipse for the java side of SAGA and swi prolog for implementing the planner. Unity3D and visual studio is used developing the Unity 3D environment. Google drive is used for storing our reports and spreadsheets.

4 Work plan

4.1 Introduction

In this chapter we outline how we expect the sprints to be divided up and what will be done in each sprint.

4.2 Sprint plan

Sprints 1:
- Study the SAGA System
- Study GOLOG
- Study Unity
- Read research material

Sprints 2-4:
- Research planner improvements
- Implement a planner

Sprint 5:
- Implement planner improvements
- Draft research paper for planner
SAGA-Interactive Storytelling

Sprint 6:
- Research Paper on planner improvements
- Unity development preparation

Sprint 7-8:
- Unity development

Sprint 9:
- Finish paper
- Finish report
- Polishing of unity environment
- Possible streamlining of project setup
- Working on stability of our project version.

5 Risk Analysis

It is important for the team to be aware of potential risks when starting the project, in order to be aware of potential time lost and to be prepared to respond to the risks as needed. These are the main risks we see in our project:

<table>
<thead>
<tr>
<th>Description</th>
<th>Response</th>
<th>Chance</th>
<th>Impact</th>
<th>Risk Factor</th>
<th>“owner”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inability to find and implement a good planner</td>
<td>Research paper will not be made, a report outlining what was attempted will be made instead.</td>
<td>30</td>
<td>10</td>
<td>300</td>
<td>Guðni</td>
</tr>
<tr>
<td>Planner improvement requires a major change on other parts in the system</td>
<td>Consult with Product owner on whether change is feasible or moving only to unity</td>
<td>15</td>
<td>20</td>
<td>300</td>
<td>Guðni</td>
</tr>
<tr>
<td>A lot of workload in other courses</td>
<td>Attempt to make up for lost work the next week/sprint</td>
<td>15</td>
<td>5</td>
<td>75</td>
<td>Kári</td>
</tr>
</tbody>
</table>

- Sprint 2 - There was a lot of workload in other courses and we worked up the lost hours in the following sprints.

<table>
<thead>
<tr>
<th>Risk Type</th>
<th>Description</th>
<th>Chance</th>
<th>Impact</th>
<th>Risk Factor</th>
<th>“owner”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sickness</td>
<td>More work when feeling better</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>Magnús</td>
</tr>
</tbody>
</table>

- Sprint 2 - Maggi got sick but that didn’t matter since he only missed one meeting.
The largest risks by far are the ones concerning the planner since those can be detrimental to the project. The most likely one being that we won’t figure out a way to improve the planner which results in a lot of “wasted” hours and we might have to skip the research paper and replace it with a report that outlines what went wrong or why it was so difficult to make a good planner. The second large risk factor is that there is that there is another component in the SAGA system that prevents us from implementing a desired planner which would force us to either a) attempt to change the component or ask the product owner to do that for us or b) stop working on the planner and going completely into the unity side c) find a different way to improve the planner.

5.1 Occurrences

In sprint 2 Maggi got sick and had to stay in bed for 3 days, luckily Maggi didn’t miss out on much during the sick days since only one of them was a day we would regularly meet up and we spent most of the day on a meeting with the product owners so we could simply inform him on the keypoints of the meeting when he came back so very little work was was lost that day.

It was also convenient that this happened in sprint 2 since we had planned that sprint to be slow. We knew beforehand that sprint that we had a lot of big assignments in other courses so we decided to include fewer story points into that sprint.

6 Design

In a nutshell, the saga system generates stories based on an outline library and represents it as a sequence of events. It then cuts these events down into a list atomic actions and those actions are fed into the simulated world.

The simulated world will then return state changes back to the system as the stories unfold and also possible unforeseen changes caused for instance by dynamic systems in the world or a human controlled agent and the saga system will feed the next actions to be performed back to the simulated world.

6.1 GOLOG planner

A research paper with details about the GOLOG planner can be seen in the appendix. The main chapters being problem formulation, proposed approach and evaluation.
6.2. Unity3D Viking Village

Viking Village is a 3D environment made with the Unity3D game engine. This environment is the current front end of the SAGA System and is a simulated world wherein you can see the stories generated by the SAGA system transpire.

6.2.1 Initial State

When starting the project the Viking Village was in a pretty bad state. It could not simulate some of the simplest possible stories in the world, all of the characters used the same character model and only had one animation which was playing an air guitar. Additionally there were some texture and collider issues in the scene such as the fountain being walked through by the characters and the fountain water being white. There was an almost complete lack of organization of game-objects and scripts, and a high amount of useless scripts and game objects, many of which appear to have been used for testing, prototyping or functionality that was aborted, and simply left sitting in the project.

The project has a lot of circular dependencies, no code and naming conventions, no documentation, few comments and a substantial amount of hard-coded strings and magic numbers. This made finding, fixing and deleting scripts and objects hard, as side effects were hard to understand, or even detect.

6.2.2. Our goals for viking village

Our primary goal for the viking village was to make sure that a story could take place in the viking village, then make more varied characters that seem somewhat realistic. Then fixing issues in the environment such as collider and texture issues and finally cleaning up useless code and objects in the unity system.

7 Project backlog

The product backlog is a list of things that need to be done for the project. They have priorities A, B and C where all entries with priority A should ideally be done before the entries marked as B and all B’s should be done before C’s. The entries are also marked with story points which are an estimate of how much time is expected to be spent on an entry relative to the other entries.

<table>
<thead>
<tr>
<th>ID</th>
<th>User story</th>
<th>Priority</th>
<th>Story points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Draft a risk report</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Study prolog/unity</td>
<td>A</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Try to understand prolog code in SAGA</td>
<td>A</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>Task</td>
<td>Priority</td>
<td>Notes</td>
</tr>
<tr>
<td>---</td>
<td>----------------------------------------------------------------------</td>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td>4</td>
<td>Setup a timeplan</td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Setup reports</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>Research heuristic for the GOLOG planner</td>
<td>A</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>Design the GOLOG planner</td>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>Implement any basic search for planner</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>Implement a cost first planner</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>Find a heuristic for best first/A* search for the planner</td>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>Implement best first/A* search</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>Set up the SAGA system Java side</td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>Write a final release of the research paper</td>
<td>A</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>Draft a research paper for the planner</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>Write a second draft for the research paper</td>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>Connect the planner with viking village</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>Final report</td>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>18</td>
<td>Familiarize with unity code base</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>19</td>
<td>Clean up unity code base</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>Delete unwanted objects</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>Reorganize unity architecture</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>22</td>
<td>Extend unity functionality</td>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>23</td>
<td>Add animations</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>Find more variable models</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>Make sure story doesn't brake</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>26</td>
<td>Prevent revisiting states in planner</td>
<td>B</td>
<td>5</td>
</tr>
<tr>
<td>27</td>
<td>Get a better understanding of how unity works</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>28</td>
<td>Design a more interesting game for the unity side</td>
<td>B</td>
<td>13</td>
</tr>
<tr>
<td>29</td>
<td>Implement a more interesting game on the unity side</td>
<td>B</td>
<td>13</td>
</tr>
<tr>
<td>30</td>
<td>Implement a planner heuristic that makes more interesting stories</td>
<td>B</td>
<td>5</td>
</tr>
<tr>
<td>31</td>
<td>Prepare status meeting 1</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>32</td>
<td>Prepare status meeting 2</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>33</td>
<td>Prepare status meeting 3</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>34</td>
<td>Prepare final presentation</td>
<td>A</td>
<td>5</td>
</tr>
</tbody>
</table>
8 Progress report

8.1 Preparation

Before starting the sprints we had done some preparation work for the project, which included setting up a work plan, meeting up with product owner and reading documentation about the project and organization planning. This spanned from 9.January - 26.January.

8.2 Sprint 1

This sprint was 2 weeks and lasted from 27.January - 9.February.

8.2.1 Backlog

This is a preparation sprint so the stories we chosen were the one needed to be done before anything else. We estimate that each story point corresponds to roughly 5 hours.

<table>
<thead>
<tr>
<th>ID</th>
<th>UserStory</th>
<th>Priority</th>
<th>Story points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Draft a risk report</td>
<td>A</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>Study prolog/unity</td>
<td>A</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>Setup a timeplan</td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Setup reports</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>Set up the SAGA system java side</td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Total :</td>
<td></td>
<td>20.5</td>
</tr>
</tbody>
</table>

8.2.3 Burndown

Here is the burndown chart for sprint 1. Showing the ideal progress, our progress and amount of hours spent each day working on the project.
8.2.4 Retrospectives

Overall we all agreed that this sprint went well and we were happy with the progress of our understanding of the systems and programming languages required for our project. Our reports and documentation are of good quality. The biggest downside is that we spent a bit too much time on other projects when we met and have agreed to avoid this in future sprints.

8.3 Sprint 2

This sprint was 2 weeks and lasted from 10. February - 23. February.

8.3.1 Backlog

We have many large assignments in other courses so we will schedule a slow sprint. We finished 20.5 story points in the first sprint which is around 100 hours so we estimate that our storypoint to hour ratio is 1:5. Knowing we will not have as much time as we wanted in this sprint we have made a sprint backlog that gives us some space if we are running out of time. 19 points are still a lot but story 3 is a story that can be carried over to the next sprint if we still need a better understanding.

<table>
<thead>
<tr>
<th>ID</th>
<th>UserStory</th>
<th>Priority</th>
<th>Story points</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Study prolog/unity</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Try to understand prolog code in SAGA</td>
<td>A</td>
<td>13</td>
</tr>
<tr>
<td>22</td>
<td>Prepare status meeting 1</td>
<td>A</td>
<td>3</td>
</tr>
</tbody>
</table>

After meeting with product owners we found out that story 3 was overestimated. We only needed to look at a tiny portion of the prolog code in SAGA. Story 3 was 13 points but was now only 3 points. We now had a sprint backlog that contained only 3 points which was way under our plan so we decided to add two new stories into the backlog. We were able to finish them both in this sprint. According to our estimates and plans we were able to finish way more than we anticipated due to wrongly estimated stories. The story marked in red is the one we miscalculated and the green ones are those we added to the sprint backlog.

<table>
<thead>
<tr>
<th>ID</th>
<th>UserStory</th>
<th>Priority</th>
<th>Story points</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Study prolog/unity</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Try to understand prolog code in SAGA</td>
<td>A</td>
<td>13</td>
</tr>
<tr>
<td>22</td>
<td>Prepare status meeting 1</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>Implement any basic search for planner</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>Implement a cost first planner</td>
<td>A</td>
<td>5</td>
</tr>
</tbody>
</table>

Total: 29
8.3.3 Burndown
This burndown shows the progress as the sprint had 29 story points from the beginning. As shown a lot happened on 17.Feb, we had a meeting with product owners that day.

8.3.4 Retrospectives
We decided to include less story points in this sprint due to a lot of big assignments in other courses. We miscalculated the size of the main story we included in the sprint and finished that story in lot less time than we thought so we had to change the sprint backlog to get more stories into the sprint. The sprint was supposed to be a slow one but we finished a total of 29 story points which is much more than we started of with. We were really happy we were able to finish more stories than we anticipated, so we can spend more time on other things regarding the project. However we saw that some of the stories are not estimated correctly, we will look over the rest of the stores and try to see if we have made the same mistakes with other stories. We also feel that we need to meet more often than we did in this sprint.

8.4 Sprint 3
This sprint was 2 weeks and lasted from 24.February - 9. March.

8.4.1 Backlog
In this sprint we are going to focus on coming up with a heuristic for the planner. We have already made a planner that works but it is too slow, so a heuristic will hopefully increase its speed. Research heuristic for the planner is our main goal in this sprint this story is large since we have to deconstruct the stories the planner makes to find out where we can make the planner smarter. After that we can add that to the cost first search the planner is currently running.
<table>
<thead>
<tr>
<th>ID</th>
<th>UserStory</th>
<th>Priority</th>
<th>Story points</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Research heuristic for the GOLOG planner</td>
<td>A</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>Implement best first/A* search</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td></td>
<td>19</td>
</tr>
</tbody>
</table>

8.4.3 Burndown
As you can see there is a flatline at the start of the sprint altho we spent 27 hours working the first two days and that is due to a minor preparation we had to finished before starting on the planner (reading documentation and reports about prolog).

8.4.4 Retrospectives
We are pleased with the outcome of this sprint, we made a planner that performs better than the previous planner and also we met and work more together in this sprint than the sprints before this one.. We were not able to finish one story in the sprint backlog but we will continue working on that one in next sprint.

8.5 Sprint 4
This sprint will be 2 weeks and last from 10.March - 23.March.

8.5.1 Backlog
During this sprint we will continue working on the planner and hopefully this will be the last sprint working on it. We also have to prepare for status meeting 2 which is on Monday 13.March. Draft for the research paper should be finished this sprint to be handed to product owners.
8.5.2 Burndown

Here we can see that the graph flatlines over the weekdays since we mostly met to work on this project over the weekends.

8.5.3 Retrospectives

We finished the planner in this sprint and it is performing way better than the original one. There are more information about the planner and its performance in the research paper that is in the appendix. We had no setbacks in this sprint and our estimates are getting better every time.

8.6 Sprint 5

This sprint will be 2 weeks and last from 24.March - 6.April. Before we started on this project we knew that this sprint would be a tricky one, since it spanned the last week of our 12 week period and the first week of the final exams. We had final projects in each of the classes that had to deliver before 1.April so the first week of this sprint was consumed by those projects. Week 2 was then spent on exam preparation, Kári and Magnús had exams in that week. Knowing this we did not take any tasks into the sprint since we knew we wouldn’t get much work done. We had a meeting with product owners at 24.March and went over some things that were uncertain and that was the only day we worked on the final project in this sprint.
8.7 Sprint 6

This sprint will be 2 weeks and last from 7.April - 20.April.

8.7.1 Backlog

In this sprint we continued writing the research paper for the planner improvements. Alongside that we started our work in the Unity3D side of the project. We had a meeting with product owners at the start of the sprint and they told us that we had permission to start scrap most of what had been done earlier and start again with focus on better design.

<table>
<thead>
<tr>
<th>ID</th>
<th>UserStory</th>
<th>Priority</th>
<th>Story points</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Write a second draft for the research paper</td>
<td>A</td>
<td>9</td>
</tr>
<tr>
<td>18</td>
<td>Familiarize with unity code base</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>19</td>
<td>Clean up unity code base</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>21</td>
<td>Get a better understanding of how unity works</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Total :</td>
<td></td>
<td>21</td>
</tr>
</tbody>
</table>

8.7.2 Burndown

Here the graph constantly goes down from when we started working on 10.Apr.

8.7.3 Retrospectives

We were happy how we bounced back after having a tough 2 weeks before. We only had to focus on one thing after we finished our exams so everything was smooth. We were not able
to finish story number 19 but that is because we underestimated how much refactoring would be needed.

### 8.8 Sprint 7

This sprint will be 1 week and last from 21.April - 27.April.

#### 8.8.1 Backlog

This was the first sprint where we had all focus was on this project. We had changed our sprint timeline and are now working in one week sprints. We could not finish one story (19) in last sprint so we will continue working on that one in this sprint. We ended up dropping story 21 since we underestimated that task and it would be a whole final project to design and organize the Unity3D architecture from ground up. We decided to work with the code that we were provided and make it work with the planner.

<table>
<thead>
<tr>
<th>ID</th>
<th>UserStory</th>
<th>Priority</th>
<th>Story points</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>Connect the planner with viking village</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>Clean up unity code base</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>Delete unwanted objects</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>Reorganize unity architecture</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>23</td>
<td>Add animations</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>Find more variable models</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td><strong>Total:</strong></td>
<td></td>
<td><strong>22</strong></td>
</tr>
</tbody>
</table>

#### 8.8.2 Burndown

We showed up for work every day now and the progress shows that we are constantly working on the project.
8.8.3 Retrospectives
This sprint was a great success, we were able to get Unity3D to run a story from the planner so we can showcase that. We had some minor hiccups with the existing code which complicated the work a little, however everything worked out. We shared equal amount of hours and worked well as a group. We don’t want to change anything about how we work as a team.

8.9 Sprint 8
This sprint will be 1 week and last from 28.April - 4.May.

8.9.1 Backlog
In this sprint we are going to finish the research paper for the planner, prepare for the final status meeting that is on 3.May and work on finalizing the unity side of viking village. This is the second-to-last sprint so we want to be able to return the project after this sprint, if not we have one more sprint to clean things up.

<table>
<thead>
<tr>
<th>ID</th>
<th>UserStory</th>
<th>Priority</th>
<th>Story points</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Write a final release of the research paper</td>
<td>A</td>
<td>13</td>
</tr>
<tr>
<td>22</td>
<td>Extend unity functionality</td>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>33</td>
<td>Prepare status meeting 3</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Total :</td>
<td></td>
<td>26</td>
</tr>
</tbody>
</table>

8.9.2 Burndown
The progress here was not as good as we wanted since we had to spent more time on preparing the status meeting.
8.9.3 Retrospectives

We spent way more time working on the presentation for the status meeting than we estimated. So other things had to be put on hold, we were not as organized in this sprint as the previous ones. We however were pleased with how it came together at the end. We were not able to finish the research paper as we planned but we are very close so we will finish that in sprint 9. We only have one sprint left and we are confident everything will be in top shape when we turn in our project.

8.10 Sprint 9

This sprint will be 1 week and last from 5.May - 11.May.

8.10.1 Backlog

This is the last sprint of the project and these are the last stories in our backlog with priority A. Extend unity functionality is a combination of adding new features and organizing the structure of the unity code. So story 21 that we dropped in sprint 6 is sort of being combined with story 22, we started rewrite on the unity system and made a small design document outlining what was done and what was left to rewrite.

<table>
<thead>
<tr>
<th>ID</th>
<th>UserStory</th>
<th>Priority</th>
<th>Story points</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Write a final release of the research paper</td>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>17</td>
<td>Final report</td>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>22</td>
<td>Extend unity functionality</td>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>34</td>
<td>Prepare final presentation</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td><strong>29</strong></td>
</tr>
</tbody>
</table>

8.10.2 Burndown

We showed up each day of the sprint and the progress was according to that.
8.10.3 Retrospectives

We finished each of these stories and the project as a whole. This last sprint was successful and we feel like we delivered a good product.

8.11 Project status

8.11.1 Hours worked

Here is the ratio of hours worked divided between us 3. We all surpassed the 300 hour limit.

8.11.2 Burndown

This is the project burndown, as we can see we started slow but after the 12 week period was over we were able to put in more hours work up the time we lost earlier.

8.11.3 Summary

We have finished our planner improvements and a research paper explaining what we did. The Unity3D side of the project is finished and we are now able to showcase the planner functionality. We started a rewrite for the Unity3D side which is in a good place. It is not complete, as it was not a priority of the project to make a complete rewrite of the system. It is designed with that in mind that extending its functionality should be easier than continuing with the old Viking Village and has documentation, which future developers can find in the git repository, describing how it works and suggestions for future work.
Appendix
Saga Interactive Storytelling

David Thue, Stephan Schiffel, Guðni Fannar Kristjánsson, Kári Eiríksson, Magnús Vilhelm Björnsson
School of Computer Science, Reykjavik University
Menntavegur 1, Reykjavik, 101, Iceland
{davidthue, stephans, gudni14, kari14, magnusvb14@ru.is}

Abstract
Interactive storytelling is a form of entertainment where the storyline is not determined before hand and can be changed during runtime depending on how the story is played. Heuris-tic can be useful when dealing with large action spaces and a distinction needs to be made between good and bad actions. Interactive systems needs to make plans at runtime and therefore have to be fast. Using heuristics and other method to reduce the state space then become necessary to keep the speed up. This paper describes an approach for using heuristic function to determine what actions are viable when building a story in a interactive environment. We will describe ways to reduce search space before adding heuristics to the system and also how a heuristic value can be calculated using only the facts known about the environment. To be able to make use of the facts about the interactive world we have to make sure that the representation of the world is always the same if not the heuristic will not be accurate. By using heuristic to guide what action is taken next we should be able to make up a plan at runtime.

Introduction
Compared to other kinds of entertainment, computer games are unique in offering players a wide variety of opportunities to influence what happens next. Meanwhile, stories are usually pre-defined: no matter what the audience does, the same sequence of events occurs. When stories are put into games, however, players can influence the game in ways that disrupt a pre-defined story, resulting in a story that no longer makes sense. The SAGA project (Thue and Halldórsson 2015) combines different techniques in Artificial Intelligence (AI) to manage this problem by automatically simulating and adapting stories while the game is underway. Since Fall 2014, many students have been implementing the SAGA project as a software system; this system can currently generate stories made up of individual characters’ actions and display them in a simple Unity3D environment. The SAGA project uses a planner among other techniques to generate stories. Originally the planner was written as a PDDL (McDermott et al. 1998) planner but was later on exchanged for a GOLOG (Levesque et al. 1997) planner because PDDL operates under closed world assumption (everything about the world is known) but GOLOG does not which can be beneficial for the expressiveness of the planner. (Gudmundsson 2017) The initial implementation of the GOLOG planner was inefficient. In this paper we will discuss some of the problems regarding the planner and some improvements made to the planner.

Problem Formulation
SAGA operates such that there is a designer that designs a story outline. Each outline contains 3 things, an initial state of the world which must be true before the story can be performed, an intermediate goal which must be true at some point of the story and a final goal which must be true at the end of the story.

Each story has some entities representing objects in the world such as characters or articles. These entities are then of 2 types: abstract and concrete. concrete entities are simply entities that exist in the world (e.g. john, sarah, gold_ring, etc). Abstract entities are essentially roles for the concrete entities that are defined in the outline but are not decided until the story is underway. An example of an abstract character could for example be hero which could be mapped to the concrete characters john or sarah so that they would have the role of the hero in the story.

A complete story is then a sequence of story actions that are performed from the initial state that at some point achieves the intermediate goal and satisfies the final goal when all actions have been performed. An action is something that can change the state of the world; actions have some preconditions and some effects. An example action could be steal(john, sarah, gold_ring) which has the precondition has(sarah, gold_ring) and the effects ~has(sarah, gold_ring) and has(john, gold_ring). In addition to these regular actions there are special mapping actions which map concrete characters to abstract characters. They have no effect on the world, but are necessary to determine whether an outline has been completed or not.

Now to get the whole picture we can consider this simple story outline:

- Initial state:
  character(john, concrete)
  character(sarah, concrete)
  character(sam, concrete)
  character(heroe, abstract)
character(mother, abstract)
character(villain, abstract)
article(gold_ring, concrete)
article(heirloom, abstract)
has(sarah, gold_ring)
parentof(sarah, john)
has(mother, heirloom)
parentof(mother, hero)

* Intermediate goal: has(villain, heirloom)
* Final goal: has(mother, heirloom)

Then an example of a valid story could be something like:
1. map(hero, john)
2. map(mother, sarah)
3. map(villain, sam)
4. map(heirloom, gold_ring)
5. steal(villain, mother, heirloom)
6. steal(hero, villain, heirloom)
7. give(hero, mother, heirloom)

This story describes a villain(sam) which steals a heirloom(gold_ring) from a mother(sarah) which is then stolen back by the hero(john) and returned to the mother. The intermediate goal is met at action 5 and the final goal condition is met at action 7.

Originally the problem was solved by a PDDL-planner, however PDDL planners operate under closed world assumption so that everything in the world is either stated as true or assumed to be false. This had the effect that story outlines could only contain predicates that the designer wanted to be true because stating that something is false is meaningless because it was assumed to be false anyways. As an example the PDDL-planner could handle having the intermediate goal has(villain, heirloom) but not the intermediate goal ¬has(villain, heirloom).

Interactive storytelling(IS) systems build a story during runtime and therefore have to be quick and clever when searching for valid sequence of actions. For this to be possible we need to have heuristic that helps the system determine what action paths are clever to take at any given time.

The GOLOG planner that we extended was not using heuristic function to help with the decision making. Since it was not using any heuristic it simply did a blind search when planning a story. It ran blind iterative deepening search algorithm so it was looking at every possible combinations of stories until it found a satisfying plan both good and bad ones. Our contribution is implementing a heuristic function that can help the planner avoid spending unnecessary time considering bad plans.

**Related Work**

Many IS systems have been made using heuristic search to minimize the search space. In 2002 Marc Cavazza et al(Cavazza, Charles, and Mead 2002) presented a character-based IS system that supports anytime intervention by the spectator inside a 3D environment. The heuristic function calculates for each sub-task so forward search can use these values to select sub-tasks. This solution is similar to what we are trying to do, interact with a story that is being simulated and disrupt the plan that has been made for the story. However it differs from our problem because each character is assigned a role or story to act out, but we don’t want to assign specific roles to specific characters, but rather have the roles assigned at runtime and potentially changed during runtime under some conditions (e.g. if the hero is killed by a player find a new hero).

Another approach for using heuristic search to solve a planning problem is the Heuristic Search Planning(HSP)(Cavazza et al. 2007) presented by Marc Cavazza et al. There each character has its own HSP to determine actions, this uses a STRIPS-like description(Nilsson and Fikes 1970) for operators. The HSP is used in a real-time environment and uses a real-time A* search to create a dynamic plan. The heuristic value is computed with a value iteration method as the contribution of a operator to reach the final goal. This approach does not work for us since our systems has no proper states, only an initial state along with sequence of actions. STRIPS model solves a relaxed version of the planning where all actions are taken at the same time in order to estimate the distance to the goal state. But the representation we are using in GOLOG makes this step impossible since we cannot take all actions because the state representation is a sequence of actions.

**Proposed Approach**

Planning problems such as this typically use an informed search method like best-first search or A* search, both of these search methods require a heuristic function to estimate which plans are likely to be good. Additionally those search methods can end up searching through the same state more than once so it is important to attempt to minimize such repetitions.

**State Space Reduction**

When searching blindly through the state space it was noticeable that some equivalent stories were being considered, so the first step in improving the planner was to try to minimize the amount of repeated or equivalent states being considered.

**Equivalent actions** The first such example can be seen in these 2 equivalent stories:

\[
\left[ \text{map(} \text{hero, } \text{john} \text{)}, \text{find(} \text{hero, } \text{necklace} \text{)} \right]
\]

\[
\left[ \text{map(} \text{hero, } \text{john} \text{)}, \text{find(} \text{john, } \text{necklace} \text{)} \right]
\]

in this case we have 2 different representations of the same story since john and hero are mapped to be the same character then the find action for each of them will result in the exact same thing. To fix this all actions were altered so that abstract characters such as the hero could not perform any actions, because every abstract character will eventually be mapped to a concrete character so it is sufficient that actions can only operate on concrete entities.
Independent actions  It was noticeable that some actions could be performed in any order while still resulting in the same state of the world. for example:

\[
\begin{align*}
&[\text{find}(\text{sarah}, \text{necklace}), \text{find}(\text{john}, \text{ring})] \\
&[\text{find}(\text{john}, \text{ring}), \text{find}(\text{sarah}, \text{necklace})]
\end{align*}
\]

are the exact same stories in the sense that they result in the exact same state of the world. This is because the two actions share no entities which means one of them can be executed regardless of whether the other one has been executed so they are essentially independent of each other. Additionally all mapping actions are always independent of each other. Reducing the state space using these facts was done by finding an absolute ordering (e.g. alphabetical ordering) for these independent actions. That way any new action added to the state would be moved according to this absolute ordering until it was adjacent to an action that it was not independent from, then simply discarding it if that sequence of actions had been seen before. For example if we have entities denoted by the letters from from A to K and the action sequence \(a(A, B), a(E, F), a(G, H), a(J, K)\) where \(a\) is an arbitrary action then adding a new action \(a(C, D)\) using the old version would simply append the action \(a(C, D)\) like so:

\[a(A, B), a(E, F), a(G, H), a(J, K), a(C, D)\]

while using this absolute ordering it will be moved left until either a non-independent action is found or if the action to the left is a lower value in the given ordering (in this case alphabetical ordering) so this would be the sequence of actions:

\[a(A, B), a(C, D), a(E, F), a(G, H), a(J, K)\]

so this will be the only permutation of these actions that the planner will ever use rather than the 5! possible permutations.

Heuristic
The purpose of a heuristic is to focus the search so that plans that are likely to succeed will be considered first. In this case the heuristic is set up such that the value obtained from the heuristic function is the cost of the plan, so a lower value is better.

What we originally wanted to do was to solve this problem using a planning graph (Blum and Furst 1995). Planning graph always returns the shortest possible plan if there exists a plan. A planning graph works such that all legal actions are taken at the same time while ignoring negative effects, this is then repeated until the goal conditions are met and the number of those repetitions would be the heuristic value. However this proved really difficult with the current world representation since each state is represented as a sequence of actions and then this “take all legal” actions step doesn’t make sense since the world representations is dependent on the sequence in which the actions are taken. So another approach would be needed.

Since doing anything akin to the planning graph proved to be difficult the best approach seemed to be taking some relatively simple heuristic values and then linearly combine them into a single heuristic function, that is adding them all up with weight to each value.

Heuristic value L This heuristic value is really simple, given a plan \(P\) the value of \(L\) is calculated by simply counting the number of actions in \(P\) i.e.

\[L = \sum_{a \in P} 1\]

where \(a\) is an action in \(P\).

Heuristic value F This heuristic value is probably the most important one, it determines how far a plan is from fulfilling the final goal. For this we need to use a function called \(k\)True that is provided by GOLOG, \(k\)True works such that given a predicate \(x\) and a plan \(y\) then \(k\)True\((x, y)\) yields true if the predicate \(x\) is known true after performing the sequence of actions in the plan \(y\) from the initial state. Then the heuristic value \(F\) can be determined by the following formula:

\[F = \sum_{p \in P_p} \begin{cases} 0, & \text{if } k\text{True}(p, \text{Plan}) \\ 1, & \text{otherwise} \end{cases}\]

where \(P_{P}\) is the set of preconditions that must be true for the final goal to be achieved.

Heuristic value I This value is very similar to heuristic value \(F\), it is calculated by the formula

\[I = \sum_{p \in P_I} \begin{cases} 0, & \text{if } k\text{True}(p, \text{Plan}) \\ 1, & \text{otherwise} \end{cases}\]

where \(P_{I}\) is the set of preconditions for the Intermediate goal. Additionally for any Plan \(X\) with the action sequence \(A_1, A_2, \cdots, A_{n-1}, A_n\) if there exists an action plan \(Y\) such that \(A_1, A_2, \cdots, A_k, A_k\) where \(k < n\) and \(I = 0\) for the plan \(Y\) then \(I = 0\) for \(X\) as well. This is due to the fact that the intermediate goal can be completed at any point in the story, and once completed should be completely ignored.

Heuristic value C Each story outline has some initial conditions that must hold for the story to be executable, however not all of these initial conditions can be known beforehand. For example: \(\text{init}(\text{motherof}(\text{mother}, \text{hero}))\) this initial condition states that the abstract character mother needs to be the mother of the hero, however the truth of this statement depends on the mapping of these 2 abstract characters. If there is plan containing the mapping action \(\text{map}(\text{mother}, \text{sarah})\), but \(\text{motherof}(\text{sarah}, X)\) is false for all values of \(X\) then this is a bad plan and should get
The average runtime is shown in milliseconds. The first row in the table is simply the original planner (including the equivalence state space reduction) and is then sped up when higher weights are set to the preconditions of the intermediate goals. Test6 is then not solved until the weights of the final goal conditions are set higher but this change causes Test5 to slow down a bit.

It is also notable that the original planner is slower than the planner with the length heuristic only, this is because the length heuristic uses both of the state space reductions while the original planner can only make use of the state space reduction that prevents abstract entities to be a part of the unmapped characters.

Discussion and Future Work

Although the improvement thus far is pretty significant there still exist story outlines which cannot be completed in a reasonable amount of time. The first such case is for an outline with final goals such as this \(\text{has(hero, heirloom)}, \text{has(villain, treasure)}.\) It is fairly obvious that if characters and articles are mapped such that \(\text{hero} \neq \text{villain}\) and \(\text{heirloom} = \text{treasure}\) that the final goals of the story can never be achieved simultaneously so all stories with such mappings should be discarded. However the opposite can happen with the current heuristic, for example if there is an intermediate goals such as \(\text{has(mother, heirloom)}, \text{has(mother, treasure)}\) then mapping \(\text{treasure} \land \text{heirloom}\) to the same item would be a good mapping. This happens because the abstract character \(\text{mother}\) now only has to find one object instead of two which the heuristic considers good, when in this case it would be really bad mapping. There are at least 2 ways in which this could be solved either a better heuristic which would backtrack earlier from this bad mapping or by doing some kind of constraint solving before hand which would determine mappings that cannot be possible for the given outline (there is already a constraint solver that helps prune the search space, however it is very lenient and does not detect cases such as this).

In the current state representation all relations that are possible need to be stated to be true or false unless they are unknown in the initial state. This can be problematic when there are more than a few characters or articles in the environment. For example if we consider the \(\text{parentof(char1, char2)}\) then for 20 characters there will

---

**Initial State**

- \(\text{has(john, ring)}\)
- \(\text{hates(sarah, joe)}\)
- \(\text{hates(sarah, john)}\)
- \(\text{likes(joe, sarah)}\)
- \(\text{hates(villain, hero)}\)
- \(\text{has(hero, treasure)}\)

**Mappings**

- \(\text{map(villain, sarah)}\)
- \(\text{map(joe, hero)}\)

**Evaluation**

We prepared six test files to check how different heuristic functions performed.

- Test1 is a very simple outline with a solution of length 4.
- Test2 is the same as Test1 except the number of abstract characters was increased to 7 (up from 1).
- Test3 is the same as Test1 except concrete characters are now 10 (were 3 in Test1).
- Test4 is an outline that has 2 intermediate goals
- Test5 is a somewhat simple story with more complicated initial conditions of the abstract characters.
- Test6 is an outline with a more complicated final goal and shortest solution of length 9

Each version of the cost function was run 5 times on each test file and the average was taken, for each run the planner was given 10 seconds to complete; if it could not finish in the given time it is shown as -1 in the table, otherwise the average runtime is shown in milliseconds.

The first row in the table is simply the original planner (including the equivalence state space reduction)

As can be seen in the table checking the preconditions for the final goal massively speeds up the search for most test cases however test 4 and 6 are not completed in the given time limit and test 5 is rather slow. Test5 then becomes reasonably fast after adding the initial conditions check. Test4 is not completed until the intermediate goals are taken into consideration and is then sped up when higher weights are set to the preconditions of the intermediate goals. Test6 is then not solved until the weights of the final goal conditions are set higher but this change causes Test5 to slow down a bit.

Discussion and Future Work

Although the improvement thus far is pretty significant there still exist story outlines which cannot be completed in a reasonable amount of time. The first such case is for an outline with final goals such as this \(\text{has(hero, heirloom)}, \text{has(villain, treasure)}.\) It is fairly obvious that if characters and articles are mapped such that \(\text{hero} \neq \text{villain}\) and \(\text{heirloom} = \text{treasure}\) that the final goals of the story can never be achieved simultaneously so all stories with such mappings should be discarded. However the opposite can happen with the current heuristic, for example if there is an intermediate goals such as \(\text{has(mother, heirloom)}, \text{has(mother, treasure)}\) then mapping \(\text{treasure} \land \text{heirloom}\) to the same item would be a good mapping. This happens because the abstract character \(\text{mother}\) now only has to find one object instead of two which the heuristic considers good, when in this case it would be really bad mapping. There are at least 2 ways in which this could be solved either a better heuristic which would backtrack earlier from this bad mapping or by doing some kind of constraint solving before hand which would determine mappings that cannot be possible for the given outline (there is already a constraint solver that helps prune the search space, however it is very lenient and does not detect cases such as this).

In the current state representation all relations that are possible need to be stated to be true or false unless they are unknown in the initial state. This can be problematic when there are more than a few characters or articles in the environment. For example if we consider the \(\text{parentof(char1, char2)}\) then for 20 characters there will
Table 1: Test results for different planner heuristics shown in milliseconds, the columns L,F,I,C indicate the weight given to the corresponding heuristic value, except in the first row which is the original iterative deepening planner.

<table>
<thead>
<tr>
<th>L</th>
<th>F</th>
<th>I</th>
<th>C</th>
<th>Test1</th>
<th>Test2</th>
<th>Test3</th>
<th>Test4</th>
<th>Test5</th>
<th>Test6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1450</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1254</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>82</td>
<td>60</td>
<td>-1</td>
<td>1820</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>79</td>
<td>69</td>
<td>-1</td>
<td>91</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>90</td>
<td>69</td>
<td>269</td>
<td>93</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>13</td>
<td>96</td>
<td>43</td>
<td>128</td>
<td>99</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>11</td>
<td>82</td>
<td>43</td>
<td>127</td>
<td>184</td>
<td>95</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>18</td>
<td>74</td>
<td>47</td>
<td>133</td>
<td>184</td>
<td>109</td>
</tr>
</tbody>
</table>

need to be 400 statements of whether x is a parent of y which are all inspected by kTrue when determining whether a fact is true or false with the current sequence of actions. Because of this not only does the search space increase with the number of characters but it also becomes slower because of the increased number of relations. This is something that could be worth looking into, it could be possible to change the initial state to do some sort of aggregation, for example if john is not anyone’s parent then something along the lines of all(parentof(john, x) = false) instead making multiple parentof(john, x) = false statements.

The next steps for working on the planner could be to get it to generate more interesting stories. Even though stories planned at a decent speed there is no guarantee that the given story will be interesting, for example the story generated could be a sequence of give actions where the characters are all just giving each other items until the intermediate and final goals are met which does no make a very interesting story. It would be a lot more interesting if the characters were to steal items from each others or take them by force. A solution to this could be by setting different weights to different actions so that the give action costs 5 while steal costs 3 so steal is chosen more often than give because the steal action is more interesting than give. The cost of each action could also be increased by each time the action is taken so that after 3 give actions another give would be very costly. However the “interesting-ness” of a story is hard to determine but that work is in progress.

**Conclusion**

In this work we added heuristic search to the planner and reduced the state space that the planner had to go through which allows finish much larger stories than the original planner. There is a limit to how much a heuristic can improve the search for the planner and it can potentially get stuck in a local minimum. Additionally the speed of the kTrue function depends on the size of the initial state which can grow large with an increasing amount of entities. For future work we might have to find a way to use a different state representation that could allow us to implement a planning graph heuristic. The issue with ktrue needs to be fixed for larger stories, be it a new implementation or a fix for the current implementation.

**References**


