

Goal Orientated Action Planning Artificial Intelligence

Sam McKay Illiyan Georgiev José Díez Vlad-Eugen Tanase
Aaron Swiss-Hamlet

April 24, 2015

Contents

1	Introduction	2
2	Project Brief	3
3	Aims and Objectives	4
4	Project Management	5
5	Methodology	6
5.0.1	Design Methodology	6
5.0.2	Development Philosophy	7
5.0.3	Code Structure	7
5.0.4	Navigation	7
5.0.5	Simple and Fast Multimedia Library	7
5.0.6	Map Generation	7
5.0.7	STRIPS	8
6	Discussion and Conclusion	9
A	Appendix	11
A.1	11
A.1.1	11
A.1.2	12
A.2	Code samples	13
A.2.1	19
A.3	Literature Review	20
A.4	Final Presentation Slides	28

Chapter 1

Introduction

Project 10 of the 206CDE Real World Projects, namely Goal Orientated Action Planning (GOAP) for Game Artificial Intelligence (AI) was successfully finished by this group. In this project, we will discuss the GOAP artificial intelligence type and its uses within video games. This project was complete in eight months by a team consisting of five members. Due to the majority of the team belonging to the Games Technology course, the overall look and feel of the project is similar to a two-dimensional pixel art style video-game, although no player interaction is possible.

GOAP AI refers to a simplified STRIPS-like planning architecture, designed for real-time control of various character behaviours within video games. A good example of a GOAP AI is the F.E.A.R. video-game AI, which has won many awards and is acclaimed as one of the best gaming AIs. Various programs were used in the making of this project. The coding behind everything was done using the C++ programming language. Photoshop was used to develop various textures and sprites to be used in the game-like presentation. For graphical purposes, the Simple and Fast Multimedia Library (SFML) was used, which is based on the Open Graphics Library(OpenGL). Axosoft and GitHub were used for team organisation and progress tracking.

Chapter 2

Project Brief

Our brief for this project was to write an artificial intelligence for games using Goal Orientated Action Planning techniques. We had to produce an AI agent capable of planning and executing a simple sequence of actions. This should include programming, software engineering and design as its key areas of study in order to complete the project.

To successfully carry out the project certain job roles must be filled, these are to include:

- Project Manager
- Lead Programmer
- Lead Designer
- Programmers

At the end of this project we are expecting to be able to deliver a simple game world containing an AI agent capable of planning simple sequences of actions and executing these in real time.

Chapter 3

Aims and Objectives

The main aim of this project is to fulfil the brief of creating a Goal Orientated Action Planning Artificial Intelligence agent. This agent should be able to asses its current state against its goal state. Our AI agent should then begin to generate an action plan, that it will execute in order to achieve the desired state.

- Design and Create a Survival Game
- Create an Artificial Intelligence Agent capable of making an action plan
- Have the Artificial Intelligence Agent carry out the plan

To demonstrate this we set ourselves the objective of designing and creating a survival game in which the main character is our AI agent. This world should be a desolate environment with multiple biomes and various resources all scattered about it, all of which is randomly generated to ensure our AI agent has a new challenge every time. With this world we envisioned in mind our next objective was to create a list of relevant actions for our AI to carry out. These actions should have pre and post conditions such that our AI could create multi action plans in order to complete a task such as satisfying its need for hunger. To be able to generate these plans we would need to create a planer algorithm, for this we would use similar logic to the STRIPS implementation around which the majority of our research was based. Finally our last objective is that we need to be our able to visualize our AI agent planning and carrying out these plans. To show this we would use the SFML library to graphically represent the aforementioned environment, along with a user interface to show the AI agents "thought process".

Chapter 4

Project Management

In order for this project to succeed we needed to implement an effective project management methodology. For this we chose to implement the agile project management structure commonly used within the software industry. Due to the size and length of the project it was imperative that we implemented good project management from the beginning, without it we could have struggled to complete the project in a timely, well paced fashion. By ensuring we had this project management in place from the start it meant that we could track our progress and continue to work at a reasonable pace throughout the duration of the project.

Agile project management is a highly effective method of managing long term projects, the process can be broken down into many stages. To begin with you collect all of your user stories, or feature requests, into a product backlog. These user stories are mostly generated by members of the team and define the "wish list" of features to include in the project. Once we have all our user stories we can begin selecting the features we will include in the final product into a release backlog (see Appendix 1.01). This release backlog can be broken down into various stages known as sprints, these sprints can be as long as a couple of days up to a whole month. All of these sprints come together to create the full product with each sprint generally expanding upon the work completed in the previous sprint.

In order for teams to easily implement this project management methodology, various different companies have all produced their own software to provide the agile management approach. We chose to use *Axosoft's* Scrum software, this creates a personalised website for the team to use and grants access to all of the previously discussed features. Using this we quickly created all of our user stories and proceeded to arrange these into various different sprints. We decided upon five sprints for the project: Research, Planning, Design and two programming sprints. This helped us to ensure that we kept on track as each task has an estimate for how long it will take to complete. So as work is done on each task we attach a work log to it and can compare estimated time with the actual duration of the task and use that as a measure of when we will finish the project as a whole using a burn-down chart (see Appendix 1.02).

By using Scrum we were able to monitor our progress and even predict release dates, which helped to give us something to aim for. We were also able to track who was working on what and exactly what work has been done on each section as well as what is left to be done. This meant that we didn't have two people each working on exactly the same task and prevented wasting time by doing so. Without Scrum it would have been very challenging to know exactly what stage we were all at and what needed to be done to meet the next deadline. Scrum also helped with our testing process by providing a bug tracker (see Appendix 1.03), allowing us to easily track any bugs and to test new code as it was completed.

We also made use of GitHub, a code sharing repository, in order to store and share our code amongst the team. This allowed us to organise the code and maintain version control whilst also enabling the team to work on various sections at the same time. Throughout the project weekly meetings were attended, Monday with the project supervisor. Team meetings were established by the project manager based on necessity and the availability of every member, minimum once a week. On various occasions, more than one meeting was booked within the library to work, discuss or establish new tasks.

Chapter 5

Methodology

5.0.1 Design Methodology

The design methodology has meant implementing a process whereby the entire team has had some input into the creation of the program's design, both structurally and graphically. In the initial research stage of the project, this meant looking up existing user interfaces (referred to as the UI from now) and determining what kind of art style the program would have to determine the best way to present the information to the screen. Interfaces that were looked at include the World of Warcraft system and how its pre-set setup was not favoured due to a lack of specific information and that of The Sims, where the needs of each sim are covered in the simple design of the UI. Fitting the graphical design to the needs of the project requirements, taking an approach much like The Sims seemed appropriate as this provides a simple and easy to understand system for users to see. The next aspect to approach on the design aspect is that of the world's graphical look. This needed to compliment the UI design. Due to the amount of time available for the entire project as well as the size of our team, a 2D world approach made more sense, which formed the basis of our initial strategy for approaching the project.

Once past the research and initial concept stage, the design implementations need to be coupled with the programming side of the project in the planning and prototyping stage. This meant talking as a group, with each member of our team to come up with the best solution to the design challenges presented. These include showing elements of the AI's thought processes to the screen. Furthermore, aspects of the AI itself that affect its planning and how to represent the items it has on its person to use. Approaching the thought processing aspect, there are many elements that the AI could need to do, for example, obtaining wood or needing to sleep. Therefore, as there are a large variety of different actions that the AI can perform, trying to graphically represent everything in a list is not a good method of display. Therefore the next best method would be to display its thoughts in a console statement window.

The next element to address in the planning stage is the way in which the AI will have elements that affect it displayed to the screen. This means visually representing to the user what exactly is making the AI decide to do that action. As this design aspect includes programming limitations, both sides of the project needed to come together on this segment. Too many elements, which we decided to call "weights", would make programming the AI too hard for the timeframe we had. On the hand too little would not give the AI anything to work on or react to within the environment. Therefore, we decided upon using only the core aspects of a human as the weights which we would include in the program. As such, four of these are included in the final simulation: hunger, thirst, energy and warmth. These were whittled down from a larger list of weights which were the initial criteria we were going to go with however, this makes for a much simpler AI system that fits the expected project outcomes. In order to demonstrate these weights to the user, we chose to use a coloured background that changes under certain conditions. This was favoured over using a bar that changes in size as this is an obvious visual representation that is instantly clear to viewers.

The final element of the UI was to display what the AI has on its person, whether it be food or a material used in creating a fire or shelter. This was much like the appearance of the weights on the UI as they can be used with a simple tile that sticks out from the rest of the UI around it. As evidenced in the final design proposal, there are tiles intended for inventory space with the ability to remove them easily as needed due to not interacting with any other element in the program's environment.

The final graphical design challenge is the creation of the 2D environment that the AI will work within. This requires a simple implementation of images that act as "tiles" in the world to create a randomly generated world where there are a selection of biomes across a fixed space environment. To create these biomes, an xml map generator is created using a sub program that denotes which biome is where, and the image that represents the biome on that tile will be displayed based on the programming implementation. Once the map is implemented, the graphics are placed in the world and then the program generates the world. On top of the terrain tiles that are produced, there are .png files which denote resources in the world, such as trees and stone, that are placed on

top of the terrain under conditions that the resource has to have fulfilled before it spawns in that tile. Keeping the amount of resources in the world was essentially due to the timeframe and also for the implementation of other aspects in the program on the technical side. These tiles are equal in size, being 64x64 pixels and take a uniform art style which, while simple and not professionally impressing, still presents the world in an obvious manner to viewers of the program.

5.0.2 Development Philosophy

At the beginning of the project a group decision was made to maintain modularity and independence between most major parts of the implementation. This was done in order to avoid some common pitfalls associated with working in a team like having to wait for one person to finish his work for another to begin. This philosophy also insured that each component of the final project would function individually and be at a completed state as quickly as possible. The con is that in the case that we lose a team member during development the code / work he has done may be difficult to recover and / or figure out in order to complete implementation. Due to the fact that we had only three programmers for the duration of the project coordinating between them was not that difficult.

5.0.3 Code Structure

The overall structure of the code was based on the system diagram however we did not keep strictly to it. Requirements and limitations that arose during development meant that some of the parts of that diagram became irrelevant, unnecessary or unpractical. The final implementation maintained modularity and plugging in new features was easy and almost effortless. However as the project evolved and changed implementing new functionality into the existing framework and connected components became harder. This was a result of modularity. Interfaces between each component were often defined at the moment when the component was added to the framework. In some cases the contents of the compound data types had to be changed in order to make implementing a specific interface easier.

5.0.4 Navigation

As we decided to not only implement a planner but to create a world for our AI to reside in it had to have a means to move around. The two obvious options explored were pathfinding and steering. The design of the world meant there would be no situations in which the agent will need to navigate negative spaces. Knowing this we decided to avoid pathfinding and used steering instead. A basic collision avoidance steering behavior was implemented. Based on angles it steers around obstacles which are in front of it and in its path. This was an ideal and sufficient movement mechanic for this project however research into different pathfinding techniques was conducted. As with other parts of the project the navigation underwent several changes to accommodate for alterations in the rest of the project.

5.0.5 Simple and Fast Multimedia Library

Using this OpenGL based API we insured quicker implementation of all visuals associated with the project and avoided having to develop basic features ourselves. Basic file type loaders, vector data types, canvas and view port management are just a few of the features we used which sped up creating the visuals of our project. Using this library we had to work around and comply with some of its conventions and concepts. While we did run into issues and setbacks none halted of them were too major and usually got resolved within minutes. Mostly these issues were associated with our unfamiliarity with SFML. Going through the documentation was the most common way we dealt with these. A good example of such a problem is the way SFML draws sprites on the screen. We expected this would be done from the center compensating for the size of the sprite. In reality the coordinates given would be the top left corner of the sprite. This made the collision avoidance look faulty and caused us to look for a solution within the steering code itself. As this was one of the first problems we faced it set a trend for most issues we encountered from that point on and fixing them was much easier and quicker.

5.0.6 Map Generation

For this project, a random map generator has been created. The size of the map in height and width is necessary, as well as the number of biomes to be created. Afterwards, each biome must have an individual, distinguishable character such as "#" or "*" and a limit of spaces this biome can take. The generator sets the starting position of each biome, with a minimum distance between each biome. The map starts with a default empty biome, drawn with the blue colour. After each position is set accordingly, the drawing starts from the middle, using the limit to take the spaces and checking if the spaces are not used by any other biome except the default one.

At the end of the generation, the map is shown using the SFML for graphical representation and an eXtensible Markup Language file is generated, containing the number of biomes, height and width, as well as every tile data type.

The Resource generation program imports and reads the XML file in order to determine the map and biomes associated to it. Each resource has a limit as well, and can only be assigned to one biome type. Once these are set, a random number is chosen between 0 and 3. With this number, the generation of the map will start from a specific corner of the map, such as 0 being top left, 1 being top right. Each resource is constrained to be spawned on an empty space and not have any other resource in its adjacency. A random number is generated every time a suitable position is found, and if it is 1, the resource is placed. A different XML file, containing the resource information of the map is generated at the end. With this imported into the main project, we have a randomly generated map populated with resources for the AI to use.

This was done in order to test the AI in different scenarios, in random maps with randomised resources in order to get different results, actions and plans from the AI. Even though every generation is random, some control is still available through coding if a resource or biome type should fit a certain criteria is needed, as well as the ability to modify the XML file itself to change the positions or types of any biome tile or resource.

5.0.7 STRIPS

The planning algorithm used for this project is STRIPS (see Fikes and Nilsson [1971]), which stands for *Stanford's Research Institute Problem Solver*. It is used to compute the actions that should be taken by the AI agent in the game to successfully achieve a set of goals. The goals are determined by a state machine that keeps track of the anthropomorphic properties of the AI: hunger, thirst, need for warmth, energy.

This state machine can introduce limitations into the planner; for example, it can dictate that the agent cannot move heavy objects while it is tired. All of these parameters are expressed in terms of different states that can be customised by the programmer and tailored to the specific application.

The algorithm works by accepting three pieces of information:

1. **Initial State:** A list of facts about the environment. (such as: "AI Agent is at 0,0", "AI's health is 70%", "There is wood at 100,245", etc).
2. **Actions:** Pure functions that transform the state through predicate manipulation. Each action has a name, a list of parameters, a precondition list (facts about the state which have to be true in order for the action to be applicable) and a postcondition list (facts that will be added or removed from the world state upon executing a given action).
3. **Goal State:** The desired state of the world after executing a plan generated by this algorithm.

When it's asked to solve a problem, the algorithm looks for a relevant action. The list of relevant actions is defined as those whose postconditions modify facts that are present in the goal state.

Then, the algorithm checks if the problem is *trivially satisfiable*: if no action needs to be taken for the initial state to satisfy the goal state, it returns an empty plan.

An action to be tested is then selected from the set of possible candidates. Then, a list of *ground states* is generated. A ground state is an abstraction not present in the original STRIPS paper, but is one we have had to come up with in order to implement a mathematical algorithm such as this in a computer program. Ground states are a mapping from symbolic names to reified values. For example, a ground state could be something like "from: A, to: B".

Each ground state is tested: in order to be able to apply a given operator matched with a ground state, we must first find a plan that solves its pre-conditions: STRIPS tracks back from the solution. For example, if we want to open a door we must first find a key. Therefore, in order to open a door we must first devise the plan to "find a key".

Once a valid plan is found to satisfy the pre-conditions of an operator, the algorithm checks if the result of applying this action to the current world state would result in the satisfaction of the overall goal. If this is true, then we simply need to concatenate the plan to satisfy the pre-conditions of any given action to the application of the action itself and the plan is complete. This recursive definition lends itself to an infinite loop if the desired plan does not exist, but we avoid this issue by terminating the algorithm if the list of candidate actions and ground states has been exhausted.

This plan is then passed to the game, where it is interpreted and executed one frame at a time: if the AI agent needs to move, the hot loop of the game will move it closer towards its target one iota at a time.

The code for the planner can be found in Appendix A, Code Samples.

Chapter 6

Discussion and Conclusion

Overall the workflow of this project went smoothly. A large portion of the work has been done in the early stages, resulting in a base for future endeavours. Due to a team member using the Linux Operating System, various steps were taken in order to make future merging of work much easier. As such, after the first attempt, all other programming merges were done without any problems.

Through the use of *Axosoft* the organisation of the team was made significantly easier. This was updated almost daily by the project manager, and each specific card was updated timely and finished accordingly. A burn-down chart(see appendix 1.01) was available, determining the time at which the project would be done according to our work submitted and number of hours left on the remaining tasks. Effective use of GitHub was in turn used to organise the project from a programming point of view. Commits to individual branches were made regularly (see appendix 4.01) with each new project part or feature implemented. This provided every member with the ability to download and work with the latest version of the project from anywhere, as well as synchronise with the other programmers.

SFML was chosen for graphical representations due to its ease of use and efficiency. It is based on the Open Graphics library and was easy to implement within the project and used to draw the map, resources and the AI. Visual representation was not highly necessary, yet it is useful, and allowed us to develop other various skills such as creating assets and implementing them accordingly.

Various different actions can be completed, such as gathering resources such as wood and being able to use the available resources. The AI will create a plan and then proceed to execute it. The resources are displayed via the interface to the user in the Inventory section, on the bottom left corner of the screen. Due to the nature of the project, other actions can be programmed with ease within the AI, giving it a much wider array of possible plans. Various assets are available within the project yet not used by the AI, such as a fireplace that can be built using wood and used to either cook food or generate warmth in order to satisfy one of the weights, namely the Warmth weight.

The project was successful in implementing a GOAP AI capable of planning and organising actions in order to complete a goal. The AI can be tested on various maps with different conditions in order to get different results based on the resources available to the AI. We believe that this fulfills our project brief and satisfies our aims and objectives. The planner logic can now be taken and implemented into a variety of scenarios with ease. Overall this task was well managed and mostly ran to the deadlines set by the project manager. The *Axosoft* site was well used and allowed the team to stay on top of their tasks. Overall the development process ran fairly smoothly with most issues being resolved quickly, this meant that we were able to focus upon completing our brief and goals. The Team worked very well with each other and pulled together in order to complete the overall project aims. Each team member completed their assigned work, often completing more to help progress the project. On some occasions the personal deadlines of the team were not met, in these instances certain members of the team undertook other members work so that the project would stay on track.

Bibliography

- Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189 – 208, 1971. ISSN 0004-3702. doi: [http://dx.doi.org/10.1016/0004-3702\(71\)90010-5](http://dx.doi.org/10.1016/0004-3702(71)90010-5). URL <http://www.sciencedirect.com/science/article/pii/0004370271900105>.
- Bulitko, V., Lee, G. 200-. Learning in Real-Time Search: A Unifying Framework. [pdf] Available at: <http://arxiv.org/pdf/1110.4076.pdf>; [Accessed 23 April 2015]
- Lee, W. 200-. DBA* A Real-Time Path Finding Algorithm. [pdf] Available at: <https://people.ok.ubc.ca/rlawrenc/research/> [Accessed 23 April 2015]
- Sturtevant, N., Buro, M. 200-. Partial Pathfinding Using Map Abstraction and Refinement. [pdf] Available at: < <https://skatgame.net/mburo/ps/path.pdf> > [Accessed 23 April 2015]
- Anguelov, B. 2011. Video Game Pathfinding and Improvements to Discrete Search on Grid-based Maps. [pdf] Available at: < <https://takinginitiative.files.wordpress.com/2011/05/thesis.pdf> > [Accessed 23 April 2015]
- Owens, A. 2002. Secrets from the Robocode masters : Anti – gravity movement. [online] Available at : < <http://www.ibm.com/developerworks/java/library/j-antigrav/> > [Accessed 23 April 2015]
- Sutherland, A. 2013. Vector Field Obstacle Avoidance. [online] Available at : < <http://buildnewgames.com/vector-field-collision-avoidance/> > [Accessed 23 April 2015]
- Bevilacqua, F. 2012. Understanding Steering Behaviors. [online] Available at : < <http://gamedevelopment.tutsplus.com/series/steering-behaviors--gamedev-12732> > [Accessed 23 April 2015]
- Reynolds, C. 199-. Steering Behaviors For Autonomous Characters. [online] Available at : < <http://www.gamesiteb.com/pathfinding/> > [Accessed 23 April 2015]
- Graham, R., McCabe, H., Scheridan, S. 200-. Pathfinding in Computer Games. [pdf] Available at : < <http://www.gamesiteb.com/pathfinding/> > [Accessed 23 April 2015]

Appendix A

Appendix

A.1

A.1.1



A.1.2

The screenshot shows the Axosoft Scrum dashboard. On the left, there's a sidebar with 'Organize' (Projects, Releases), 'Users & Teams' (All Users, All Teams), and 'Customers'. The main area displays a Kanban board with columns: New Request, Approved, In Progress, Ready For Testing, Testing Complete, Testing Approved, and Rejected. The 'In Progress' column has the most stories. A summary at the bottom states: Total: 13 User Stories • 64.3 Hours Worked • 2.5 Hours Remaining.

User Story ID	Title	Assigned To	Priority	Release	Spent	Status
38: GOAP	GOAP	Jose Manuel Diez	Medium	4. Programming Sprint 1	32 hrs / 12 hrs	In Progress
41: World	World	Jose Manuel Diez	Medium	4. Programming Sprint 1	6.4 hrs / 0.5 hrs	In Progress
46: Main Goal - Survive	Main Goal - Survive	GOAP	Medium	4. Programming Sprint 1	0 hrs / 0.3 hrs	In Progress
52: Node System	Node System	Sam Moray	High	4. Programming Sprint 1	12 hrs / 0 hrs	Ready For Testing
55: Planner	Planner	Jose Manuel Diez	Medium	4. Programming Sprint 1	22 hrs / 0 hrs	Ready For Testing
39: Steering	Steering	Ilian Georgiev	Medium	4. Programming Sprint 1	10 hrs / 0 hrs	Ready For Testing
42: 2D Vector Library	2D Vector Library	Ilian Georgiev	Medium	4. Programming Sprint 1	15 hrs / 0 hrs	Ready For Testing
47: Time System	Time System	Ilian Georgiev	Medium	4. Programming Sprint 1	1.3 hrs / 0 hrs	Ready For Testing
50: Draw	Draw	Vlad Tanase	Medium	4. Programming Sprint 1	0 hrs / 0 hrs	Ready For Testing
56: Interim Presentation	Interim Presentation	Sam Moray	High	4. Programming Sprint 1	3 hrs / 0 hrs	Testing Approved
58: XML Research	XML Research	Vlad Tanase	Medium	4. Programming Sprint 1	0 hrs / 0 hrs	Testing Approved

Axosoft User Stories

A.2 Code samples

```

1 #pragma once
2
3 #include <algorithm>
4 #include <iostream>
5 #include <vector>
6 #include <map>
7 #include <assert.h>
8
9 #include "goap/printable.hpp"
10 #include "goap/fact.hpp"
11 #include "goap/action.hpp"
12
13 namespace Planner {
14     class Problem: public Printable {
15         State state;
16         std::map<std::string, Action> actions;
17         Goal goal;
18
19         bool satisfies(State state, Goal g, bool weak=true, GroundState* ground=NULL) {
20             for(auto it = g.begin(); it != g.end(); it++) {
21                 // Check if this fact is in the state
22                 bool found = false;
23                 for(auto its = state.begin(); its != state.end(); its++) {
24                     if(its->name == it->name) {
25                         found = true;
26                         assert(its->params.size() == it->params.size());
27
28                         // If any of the arguments are different (and not floating), it doesn't
29                         // satisfy the goal
30                         for(int i = 0; i < its->params.size(); i++) {
31                             if(weak) {
32                                 if((its->params[i] != it->params[i]) && it->floating == false
33                                 && its->floating == false)
34                                     return false;
35                             } else {
36                                 if(ground == NULL) return false;
37
38                                 // Check with the ground state
39                                 std::vector<std::string> arguments = matchBounds(it->params,
40                                         *ground);
41                                 if(arguments.size() == 0) // No relevant argument found in
42                                     current ground, skip
43                                     continue;
44
45                                 if(its->params[i] != arguments[i]) {
46                                     //std::cout << "STRONG MATCH FAILURE: " << it->params[i]
47                                     << " != " << arguments[i] << std::endl;
48
49                                     return false;
50                                 }
51                             }
52                         }
53                     }
54
55                     return true;
56                 }
57
58                 std::vector<Action*> choose_operator(Goal g) {
59                     // TODO: Add weighting/heuristics?
60                     std::vector<Action*> result;
61
62                     for(auto it = this->actions.begin(); it != this->actions.end(); it++) {
63                         // Check if any of the postconditions in this actions are relevant to some
64                         // predicate in g
65                         for(auto itp = it->second.postconditions.begin(); itp != it->second.
66                         postconditions.end(); itp++) {
67                             for(auto itg = g.begin(); itg != g.end(); itg++) {
68                                 if(itp->name == itg->name)
69                                     result.push_back(&it->second);
70
71             }
72
73         }
74
75     }
76
77     void print() const {
78         std::cout << "State: " << state;
79         std::cout << "Actions: ";
80         for(const auto& action : actions) {
81             std::cout << action->name << " ";
82         }
83         std::cout << std::endl;
84     }
85
86     void add_action(Action* action) {
87         actions.insert(action);
88     }
89
90     void remove_action(Action* action) {
91         actions.erase(action);
92     }
93
94     void set_state(State state) {
95         this->state = state;
96     }
97
98     void set_goal(Goal goal) {
99         this->goal = goal;
100    }
101
102    void print_params() const {
103        std::cout << "Params: ";
104        for(const auto& action : actions) {
105            std::cout << action->name << " ";
106        }
107        std::cout << std::endl;
108    }
109
110    void print_postconditions() const {
111        std::cout << "Postconditions: ";
112        for(const auto& action : actions) {
113            std::cout << action->name << " ";
114        }
115        std::cout << std::endl;
116    }
117
118    void print_preconditions() const {
119        std::cout << "Preconditions: ";
120        for(const auto& action : actions) {
121            std::cout << action->name << " ";
122        }
123        std::cout << std::endl;
124    }
125
126    void print_name() const {
127        std::cout << "Name: " << name;
128    }
129
130    void print_type() const {
131        std::cout << "Type: " << type;
132    }
133
134    void print_params() const {
135        std::cout << "Params: ";
136        for(const auto& action : actions) {
137            std::cout << action->name << " ";
138        }
139        std::cout << std::endl;
140    }
141
142    void print_postconditions() const {
143        std::cout << "Postconditions: ";
144        for(const auto& action : actions) {
145            std::cout << action->name << " ";
146        }
147        std::cout << std::endl;
148    }
149
150    void print_preconditions() const {
151        std::cout << "Preconditions: ";
152        for(const auto& action : actions) {
153            std::cout << action->name << " ";
154        }
155        std::cout << std::endl;
156    }
157
158    void print_name() const {
159        std::cout << "Name: " << name;
160    }
161
162    void print_type() const {
163        std::cout << "Type: " << type;
164    }
165
166    void print_params() const {
167        std::cout << "Params: ";
168        for(const auto& action : actions) {
169            std::cout << action->name << " ";
170        }
171        std::cout << std::endl;
172    }
173
174    void print_postconditions() const {
175        std::cout << "Postconditions: ";
176        for(const auto& action : actions) {
177            std::cout << action->name << " ";
178        }
179        std::cout << std::endl;
180    }
181
182    void print_preconditions() const {
183        std::cout << "Preconditions: ";
184        for(const auto& action : actions) {
185            std::cout << action->name << " ";
186        }
187        std::cout << std::endl;
188    }
189
190    void print_name() const {
191        std::cout << "Name: " << name;
192    }
193
194    void print_type() const {
195        std::cout << "Type: " << type;
196    }
197
198    void print_params() const {
199        std::cout << "Params: ";
200        for(const auto& action : actions) {
201            std::cout << action->name << " ";
202        }
203        std::cout << std::endl;
204    }
205
206    void print_postconditions() const {
207        std::cout << "Postconditions: ";
208        for(const auto& action : actions) {
209            std::cout << action->name << " ";
210        }
211        std::cout << std::endl;
212    }
213
214    void print_preconditions() const {
215        std::cout << "Preconditions: ";
216        for(const auto& action : actions) {
217            std::cout << action->name << " ";
218        }
219        std::cout << std::endl;
220    }
221
222    void print_name() const {
223        std::cout << "Name: " << name;
224    }
225
226    void print_type() const {
227        std::cout << "Type: " << type;
228    }
229
230    void print_params() const {
231        std::cout << "Params: ";
232        for(const auto& action : actions) {
233            std::cout << action->name << " ";
234        }
235        std::cout << std::endl;
236    }
237
238    void print_postconditions() const {
239        std::cout << "Postconditions: ";
240        for(const auto& action : actions) {
241            std::cout << action->name << " ";
242        }
243        std::cout << std::endl;
244    }
245
246    void print_preconditions() const {
247        std::cout << "Preconditions: ";
248        for(const auto& action : actions) {
249            std::cout << action->name << " ";
250        }
251        std::cout << std::endl;
252    }
253
254    void print_name() const {
255        std::cout << "Name: " << name;
256    }
257
258    void print_type() const {
259        std::cout << "Type: " << type;
260    }
261
262    void print_params() const {
263        std::cout << "Params: ";
264        for(const auto& action : actions) {
265            std::cout << action->name << " ";
266        }
267        std::cout << std::endl;
268    }
269
270    void print_postconditions() const {
271        std::cout << "Postconditions: ";
272        for(const auto& action : actions) {
273            std::cout << action->name << " ";
274        }
275        std::cout << std::endl;
276    }
277
278    void print_preconditions() const {
279        std::cout << "Preconditions: ";
280        for(const auto& action : actions) {
281            std::cout << action->name << " ";
282        }
283        std::cout << std::endl;
284    }
285
286    void print_name() const {
287        std::cout << "Name: " << name;
288    }
289
290    void print_type() const {
291        std::cout << "Type: " << type;
292    }
293
294    void print_params() const {
295        std::cout << "Params: ";
296        for(const auto& action : actions) {
297            std::cout << action->name << " ";
298        }
299        std::cout << std::endl;
300    }
301
302    void print_postconditions() const {
303        std::cout << "Postconditions: ";
304        for(const auto& action : actions) {
305            std::cout << action->name << " ";
306        }
307        std::cout << std::endl;
308    }
309
310    void print_preconditions() const {
311        std::cout << "Preconditions: ";
312        for(const auto& action : actions) {
313            std::cout << action->name << " ";
314        }
315        std::cout << std::endl;
316    }
317
318    void print_name() const {
319        std::cout << "Name: " << name;
320    }
321
322    void print_type() const {
323        std::cout << "Type: " << type;
324    }
325
326    void print_params() const {
327        std::cout << "Params: ";
328        for(const auto& action : actions) {
329            std::cout << action->name << " ";
330        }
331        std::cout << std::endl;
332    }
333
334    void print_postconditions() const {
335        std::cout << "Postconditions: ";
336        for(const auto& action : actions) {
337            std::cout << action->name << " ";
338        }
339        std::cout << std::endl;
340    }
341
342    void print_preconditions() const {
343        std::cout << "Preconditions: ";
344        for(const auto& action : actions) {
345            std::cout << action->name << " ";
346        }
347        std::cout << std::endl;
348    }
349
350    void print_name() const {
351        std::cout << "Name: " << name;
352    }
353
354    void print_type() const {
355        std::cout << "Type: " << type;
356    }
357
358    void print_params() const {
359        std::cout << "Params: ";
360        for(const auto& action : actions) {
361            std::cout << action->name << " ";
362        }
363        std::cout << std::endl;
364    }
365
366    void print_postconditions() const {
367        std::cout << "Postconditions: ";
368        for(const auto& action : actions) {
369            std::cout << action->name << " ";
370        }
371        std::cout << std::endl;
372    }
373
374    void print_preconditions() const {
375        std::cout << "Preconditions: ";
376        for(const auto& action : actions) {
377            std::cout << action->name << " ";
378        }
379        std::cout << std::endl;
380    }
381
382    void print_name() const {
383        std::cout << "Name: " << name;
384    }
385
386    void print_type() const {
387        std::cout << "Type: " << type;
388    }
389
390    void print_params() const {
391        std::cout << "Params: ";
392        for(const auto& action : actions) {
393            std::cout << action->name << " ";
394        }
395        std::cout << std::endl;
396    }
397
398    void print_postconditions() const {
399        std::cout << "Postconditions: ";
400        for(const auto& action : actions) {
401            std::cout << action->name << " ";
402        }
403        std::cout << std::endl;
404    }
405
406    void print_preconditions() const {
407        std::cout << "Preconditions: ";
408        for(const auto& action : actions) {
409            std::cout << action->name << " ";
410        }
411        std::cout << std::endl;
412    }
413
414    void print_name() const {
415        std::cout << "Name: " << name;
416    }
417
418    void print_type() const {
419        std::cout << "Type: " << type;
420    }
421
422    void print_params() const {
423        std::cout << "Params: ";
424        for(const auto& action : actions) {
425            std::cout << action->name << " ";
426        }
427        std::cout << std::endl;
428    }
429
430    void print_postconditions() const {
431        std::cout << "Postconditions: ";
432        for(const auto& action : actions) {
433            std::cout << action->name << " ";
434        }
435        std::cout << std::endl;
436    }
437
438    void print_preconditions() const {
439        std::cout << "Preconditions: ";
440        for(const auto& action : actions) {
441            std::cout << action->name << " ";
442        }
443        std::cout << std::endl;
444    }
445
446    void print_name() const {
447        std::cout << "Name: " << name;
448    }
449
450    void print_type() const {
451        std::cout << "Type: " << type;
452    }
453
454    void print_params() const {
455        std::cout << "Params: ";
456        for(const auto& action : actions) {
457            std::cout << action->name << " ";
458        }
459        std::cout << std::endl;
460    }
461
462    void print_postconditions() const {
463        std::cout << "Postconditions: ";
464        for(const auto& action : actions) {
465            std::cout << action->name << " ";
466        }
467        std::cout << std::endl;
468    }
469
470    void print_preconditions() const {
471        std::cout << "Preconditions: ";
472        for(const auto& action : actions) {
473            std::cout << action->name << " ";
474        }
475        std::cout << std::endl;
476    }
477
478    void print_name() const {
479        std::cout << "Name: " << name;
480    }
481
482    void print_type() const {
483        std::cout << "Type: " << type;
484    }
485
486    void print_params() const {
487        std::cout << "Params: ";
488        for(const auto& action : actions) {
489            std::cout << action->name << " ";
490        }
491        std::cout << std::endl;
492    }
493
494    void print_postconditions() const {
495        std::cout << "Postconditions: ";
496        for(const auto& action : actions) {
497            std::cout << action->name << " ";
498        }
499        std::cout << std::endl;
500    }
501
502    void print_preconditions() const {
503        std::cout << "Preconditions: ";
504        for(const auto& action : actions) {
505            std::cout << action->name << " ";
506        }
507        std::cout << std::endl;
508    }
509
510    void print_name() const {
511        std::cout << "Name: " << name;
512    }
513
514    void print_type() const {
515        std::cout << "Type: " << type;
516    }
517
518    void print_params() const {
519        std::cout << "Params: ";
520        for(const auto& action : actions) {
521            std::cout << action->name << " ";
522        }
523        std::cout << std::endl;
524    }
525
526    void print_postconditions() const {
527        std::cout << "Postconditions: ";
528        for(const auto& action : actions) {
529            std::cout << action->name << " ";
530        }
531        std::cout << std::endl;
532    }
533
534    void print_preconditions() const {
535        std::cout << "Preconditions: ";
536        for(const auto& action : actions) {
537            std::cout << action->name << " ";
538        }
539        std::cout << std::endl;
540    }
541
542    void print_name() const {
543        std::cout << "Name: " << name;
544    }
545
546    void print_type() const {
547        std::cout << "Type: " << type;
548    }
549
550    void print_params() const {
551        std::cout << "Params: ";
552        for(const auto& action : actions) {
553            std::cout << action->name << " ";
554        }
555        std::cout << std::endl;
556    }
557
558    void print_postconditions() const {
559        std::cout << "Postconditions: ";
560        for(const auto& action : actions) {
561            std::cout << action->name << " ";
562        }
563        std::cout << std::endl;
564    }
565
566    void print_preconditions() const {
567        std::cout << "Preconditions: ";
568        for(const auto& action : actions) {
569            std::cout << action->name << " ";
570        }
571        std::cout << std::endl;
572    }
573
574    void print_name() const {
575        std::cout << "Name: " << name;
576    }
577
578    void print_type() const {
579        std::cout << "Type: " << type;
580    }
581
582    void print_params() const {
583        std::cout << "Params: ";
584        for(const auto& action : actions) {
585            std::cout << action->name << " ";
586        }
587        std::cout << std::endl;
588    }
589
590    void print_postconditions() const {
591        std::cout << "Postconditions: ";
592        for(const auto& action : actions) {
593            std::cout << action->name << " ";
594        }
595        std::cout << std::endl;
596    }
597
598    void print_preconditions() const {
599        std::cout << "Preconditions: ";
600        for(const auto& action : actions) {
601            std::cout << action->name << " ";
602        }
603        std::cout << std::endl;
604    }
605
606    void print_name() const {
607        std::cout << "Name: " << name;
608    }
609
610    void print_type() const {
611        std::cout << "Type: " << type;
612    }
613
614    void print_params() const {
615        std::cout << "Params: ";
616        for(const auto& action : actions) {
617            std::cout << action->name << " ";
618        }
619        std::cout << std::endl;
620    }
621
622    void print_postconditions() const {
623        std::cout << "Postconditions: ";
624        for(const auto& action : actions) {
625            std::cout << action->name << " ";
626        }
627        std::cout << std::endl;
628    }
629
630    void print_preconditions() const {
631        std::cout << "Preconditions: ";
632        for(const auto& action : actions) {
633            std::cout << action->name << " ";
634        }
635        std::cout << std::endl;
636    }
637
638    void print_name() const {
639        std::cout << "Name: " << name;
640    }
641
642    void print_type() const {
643        std::cout << "Type: " << type;
644    }
645
646    void print_params() const {
647        std::cout << "Params: ";
648        for(const auto& action : actions) {
649            std::cout << action->name << " ";
650        }
651        std::cout << std::endl;
652    }
653
654    void print_postconditions() const {
655        std::cout << "Postconditions: ";
656        for(const auto& action : actions) {
657            std::cout << action->name << " ";
658        }
659        std::cout << std::endl;
660    }
661
662    void print_preconditions() const {
663        std::cout << "Preconditions: ";
664        for(const auto& action : actions) {
665            std::cout << action->name << " ";
666        }
667        std::cout << std::endl;
668    }
669
670    void print_name() const {
671        std::cout << "Name: " << name;
672    }
673
674    void print_type() const {
675        std::cout << "Type: " << type;
676    }
677
678    void print_params() const {
679        std::cout << "Params: ";
680        for(const auto& action : actions) {
681            std::cout << action->name << " ";
682        }
683        std::cout << std::endl;
684    }
685
686    void print_postconditions() const {
687        std::cout << "Postconditions: ";
688        for(const auto& action : actions) {
689            std::cout << action->name << " ";
690        }
691        std::cout << std::endl;
692    }
693
694    void print_preconditions() const {
695        std::cout << "Preconditions: ";
696        for(const auto& action : actions) {
697            std::cout << action->name << " ";
698        }
699        std::cout << std::endl;
700    }
701
702    void print_name() const {
703        std::cout << "Name: " << name;
704    }
705
706    void print_type() const {
707        std::cout << "Type: " << type;
708    }
709
710    void print_params() const {
711        std::cout << "Params: ";
712        for(const auto& action : actions) {
713            std::cout << action->name << " ";
714        }
715        std::cout << std::endl;
716    }
717
718    void print_postconditions() const {
719        std::cout << "Postconditions: ";
720        for(const auto& action : actions) {
721            std::cout << action->name << " ";
722        }
723        std::cout << std::endl;
724    }
725
726    void print_preconditions() const {
727        std::cout << "Preconditions: ";
728        for(const auto& action : actions) {
729            std::cout << action->name << " ";
730        }
731        std::cout << std::endl;
732    }
733
734    void print_name() const {
735        std::cout << "Name: " << name;
736    }
737
738    void print_type() const {
739        std::cout << "Type: " << type;
740    }
741
742    void print_params() const {
743        std::cout << "Params: ";
744        for(const auto& action : actions) {
745            std::cout << action->name << " ";
746        }
747        std::cout << std::endl;
748    }
749
750    void print_postconditions() const {
751        std::cout << "Postconditions: ";
752        for(const auto& action : actions) {
753            std::cout << action->name << " ";
754        }
755        std::cout << std::endl;
756    }
757
758    void print_preconditions() const {
759        std::cout << "Preconditions: ";
760        for(const auto& action : actions) {
761            std::cout << action->name << " ";
762        }
763        std::cout << std::endl;
764    }
765
766    void print_name() const {
767        std::cout << "Name: " << name;
768    }
769
770    void print_type() const {
771        std::cout << "Type: " << type;
772    }
773
774    void print_params() const {
775        std::cout << "Params: ";
776        for(const auto& action : actions) {
777            std::cout << action->name << " ";
778        }
779        std::cout << std::endl;
780    }
781
782    void print_postconditions() const {
783        std::cout << "Postconditions: ";
784        for(const auto& action : actions) {
785            std::cout << action->name << " ";
786        }
787        std::cout << std::endl;
788    }
789
790    void print_preconditions() const {
791        std::cout << "Preconditions: ";
792        for(const auto& action : actions) {
793            std::cout << action->name << " ";
794        }
795        std::cout << std::endl;
796    }
797
798    void print_name() const {
799        std::cout << "Name: " << name;
800    }
801
802    void print_type() const {
803        std::cout << "Type: " << type;
804    }
805
806    void print_params() const {
807        std::cout << "Params: ";
808        for(const auto& action : actions) {
809            std::cout << action->name << " ";
810        }
811        std::cout << std::endl;
812    }
813
814    void print_postconditions() const {
815        std::cout << "Postconditions: ";
816        for(const auto& action : actions) {
817            std::cout << action->name << " ";
818        }
819        std::cout << std::endl;
820    }
821
822    void print_preconditions() const {
823        std::cout << "Preconditions: ";
824        for(const auto& action : actions) {
825            std::cout << action->name << " ";
826        }
827        std::cout << std::endl;
828    }
829
830    void print_name() const {
831        std::cout << "Name: " << name;
832    }
833
834    void print_type() const {
835        std::cout << "Type: " << type;
836    }
837
838    void print_params() const {
839        std::cout << "Params: ";
840        for(const auto& action : actions) {
841            std::cout << action->name << " ";
842        }
843        std::cout << std::endl;
844    }
845
846    void print_postconditions() const {
847        std::cout << "Postconditions: ";
848        for(const auto& action : actions) {
849            std::cout << action->name << " ";
850        }
851        std::cout << std::endl;
852    }
853
854    void print_preconditions() const {
855        std::cout << "Preconditions: ";
856        for(const auto& action : actions) {
857            std::cout << action->name << " ";
858        }
859        std::cout << std::endl;
860    }
861
862    void print_name() const {
863        std::cout << "Name: " << name;
864    }
865
866    void print_type() const {
867        std::cout << "Type: " << type;
868    }
869
870    void print_params() const {
871        std::cout << "Params: ";
872        for(const auto& action : actions) {
873            std::cout << action->name << " ";
874        }
875        std::cout << std::endl;
876    }
877
878    void print_postconditions() const {
879        std::cout << "Postconditions: ";
880        for(const auto& action : actions) {
881            std::cout << action->name << " ";
882        }
883        std::cout << std::endl;
884    }
885
886    void print_preconditions() const {
887        std::cout << "Preconditions: ";
888        for(const auto& action : actions) {
889            std::cout << action->name << " ";
890        }
891        std::cout << std::endl;
892    }
893
894    void print_name() const {
895        std::cout << "Name: " << name;
896    }
897
898    void print_type() const {
899        std::cout << "Type: " << type;
900    }
901
902    void print_params() const {
903        std::cout << "Params: ";
904        for(const auto& action : actions) {
905            std::cout << action->name << " ";
906        }
907        std::cout << std::endl;
908    }
909
910    void print_postconditions() const {
911        std::cout << "Postconditions: ";
912        for(const auto& action : actions) {
913            std::cout << action->name << " ";
914        }
915        std::cout << std::endl;
916    }
917
918    void print_preconditions() const {
919        std::cout << "Preconditions: ";
920        for(const auto& action : actions) {
921            std::cout << action->name << " ";
922        }
923        std::cout << std::endl;
924    }
925
926    void print_name() const {
927        std::cout << "Name: " << name;
928    }
929
930    void print_type() const {
931        std::cout << "Type: " << type;
932    }
933
934    void print_params() const {
935        std::cout << "Params: ";
936        for(const auto& action : actions) {
937            std::cout << action->name << " ";
938        }
939        std::cout << std::endl;
940    }
941
942    void print_postconditions() const {
943        std::cout << "Postconditions: ";
944        for(const auto& action : actions) {
945            std::cout << action->name << " ";
946        }
947        std::cout << std::endl;
948    }
949
950    void print_preconditions() const {
951        std::cout << "Preconditions: ";
952        for(const auto& action : actions) {
953            std::cout << action->name << " ";
954        }
955        std::cout << std::endl;
956    }
957
958    void print_name() const {
959        std::cout << "Name: " << name;
960    }
961
962    void print_type() const {
963        std::cout << "Type: " << type;
964    }
965
966    void print_params() const {
967        std::cout << "Params: ";
968        for(const auto& action : actions) {
969            std::cout << action->name << " ";
970        }
971        std::cout << std::endl;
972    }
973
974    void print_postconditions() const
```

```

                }
69
    }
71
}
73     return result;
75
76 std::vector<Fact> update_state(State s, Plan p, GroundState ground) {
77     for(auto it = p.begin(); it != p.end(); it++) {
78         // First, find an action with the same name as this fact.
79         auto act_it = this->actions.find((*it).name);
80
81         // If no action with a matching name is found, ignore this step.
82         if(act_it == this->actions.end())
83             continue;
84
85         // Otherwise, apply it to the state.
86         Action act = act_it->second;
87         s = act.engage(s, ground);
88     }
89
90     return s;
91 }
92
93 std::vector<GroundState> generateGroundStates(Action* act, State s, Goal g) {
94     std::vector<GroundState> result;
95     std::vector<std::string> values;
96
97     // Extract all values
98     for(auto it = s.begin(); it != s.end(); it++) {
99         for(auto itp = it->params.begin(); itp != it->params.end(); itp++) {
100            values.push_back(*itp);
101        }
102    }
103    for(auto it = g.begin(); it != g.end(); it++) {
104        for(auto itp = it->params.begin(); itp != it->params.end(); itp++) {
105            values.push_back(*itp);
106        }
107    }
108
109    std::sort(values.begin(), values.end()); // don't ask
110
111    do {
112        int ctr = 0;
113        GroundState target;
114
115        // Find all combinations
116        for(auto itp = act->params.begin(); itp != act->params.end(); itp++) {
117            //std::cout << *itp << " " << values[ctr] << std::endl;
118            target[*itp] = values[ctr++];
119        }
120
121        result.push_back(target);
122    } while(std::next_permutation(values.begin(), values.end()));
123
124    return result;
125 }
126
127 Plan* solve_(State s, Goal g, int depth = 0, GroundState* stepGround = NULL) {
128     std::string padding;
129     Plan* p = new Plan;
130
131     for(int i = 0; i < depth; i++) {
132         padding += " ";
133     }
134
135     std::cout << padding << "Subproblem state: " << std::endl;
136     for(auto it = s.begin(); it != s.end(); it++) {
137         std::cout << padding << " - ";
138         it->print();
139         std::cout << std::endl;
140     }
141
142     std::cout << padding << "Subgoal: " << std::endl;
143     for(auto it = g.begin(); it != g.end(); it++) {

```

```

145         std::cout << padding << " - ";
146         it->print();
147         std::cout << std::endl;
148     }
149
150     if (stepGround != NULL) {
151         std::cout << padding << "Ground: " << std::endl;
152         for (auto it = stepGround->begin(); it != stepGround->end(); it++) {
153             std::cout << padding << " - " << it->first << ":" << it->second;
154             std::cout << std::endl;
155         }
156     }
157
158     // Vector of applicable operators.
159     auto candidate_operators = choose_operator(g);
160
161     while(true) {
162         bool weakMatch = stepGround == NULL;
163
164         if (satisfies(s, g, weakMatch, stepGround)) {
165             std::cout << padding << "SOLUTION << state: " << std::endl;
166             for (auto it = s.begin(); it != s.end(); it++) {
167                 std::cout << padding << " - ";
168                 it->print();
169                 std::cout << std::endl;
170             }
171
172             // Mission accomplished! \o/
173             return p;
174         }
175
176         if (candidate_operators.size() == 0) {
177             return NULL;
178         }
179
180         // Get the first candidate operator, and pop it from the list
181         auto candidate = candidate_operators[0];
182         candidate_operators.erase(candidate_operators.begin());
183
184         // Generate possible ground states
185         std::vector<GroundState> grounds = this->generateGroundStates(candidate, s, g)
186 ;
187         auto ground = grounds.end();
188         Plan* subplan = NULL;
189         bool found = false;
190
191         for (auto it = grounds.begin(); it != grounds.end(); it++) {
192             ground = it;
193             GroundState* copiedGround = &(*it);
194
195             // Get a plan that solves candidate's preconditions with the current
196             // ground candidate
197             subplan = this->solve_(s, candidate->preconditions, depth + 1,
198             copiedGround);
199             if (subplan == NULL) {
200                 //std::cout << "Subplan invalid, continuing to next ground." << std::endl;
201                 continue;
202             }
203
204             // Check if this ground state satisfies the subgoal.
205             // Run all actions in the subplan and update h
206             State h = update_state(s, *subplan, *ground);
207             // Run the current action candidate and modify the state
208             h = candidate->engage(h, *ground);
209
210             if (satisfies(h, g, weakMatch, copiedGround)) {
211                 found = true;
212                 s = h;
213                 break;
214             } else {
215                 std::cout << "Nope, invalid final state." << std::endl;
216             }
217         }
218
219         if (found == false) // None of the ground states are valid: no solution

```

```

217     return NULL;
218
219 // Add all steps in subplan to the Master Plan
220 for(auto it = subplan->begin(); it != subplan->end(); it++) {
221     p->push_back(*it);
222 }
223
224 // Add the current action to the plan, matching params to ground.
225 std::vector<std::string> arguments = matchGrounds(candidate->params, *ground);
226 p->push_back(Step(candidate->name, arguments)); // Housekeeping
227 }
228
229 public:
230 Problem(std::vector<Action> _actions, State _state, Goal _goal):
231     state(_state), goal(_goal) {
232
233     for(auto it = _actions.begin(); it != _actions.end(); it++) {
234         this->actions.insert(std::pair<std::string, Action>(it->name, *it)); // Thanks
235     }
236 };
237
238 void print() {
239     std::cout << "Available actions: " << std::endl;
240     for(auto it = this->actions.begin(); it != this->actions.end(); it++) {
241         std::cout << " - ";
242         it->second.print();
243         std::cout << std::endl;
244     }
245     std::cout << std::endl;
246
247     std::cout << "Problem state: " << std::endl;
248     for(auto it = this->state.begin(); it != this->state.end(); it++) {
249         std::cout << " - ";
250         it->print();
251         std::cout << std::endl;
252     }
253     std::cout << std::endl;
254
255     std::cout << "Goal: " << std::endl;
256     for(auto it = this->goal.begin(); it != this->goal.end(); it++) {
257         std::cout << " - ";
258         it->print();
259         std::cout << std::endl;
260     }
261     std::cout << std::endl;
262 }
263
264 Plan* solve() {
265     return this->solve_(this->state, this->goal);
266 }
267
268 };
269

```

codesamples/problem.hpp

```
1 #pragma once
2
3 #include <iostream>
4 #include <vector>
5 #include <map>
6 #include <assert.h>
7
8 #include "goap/printable.hpp"
9 #include "goap/fact.hpp"
10
11 namespace Planner {
12     class Action : public Printable {
13         public:
14             std::string name;
15             std::vector<std::string> params;
16             std::vector<Fact> preconditions;
17             std::vector<Fact> postconditions;
18 }
```

```

19     Action(std::string _name, std::vector<std::string> _params, std::vector<Fact>
20     _preconditions, std::vector<Fact> _postconditions):
21         name(_name), params(_params), preconditions(_preconditions), postconditions(
22         _postconditions) {};
23
24     void print() {
25         std::cout << this->name << "(";
26         for(auto it = this->params.begin(); it != this->params.end(); it++) {
27             std::cout << *it << ", ";
28         }
29         std::cout << ")";
30
31         std::cout << ", preconditions: ";
32         for(auto it = this->preconditions.begin(); it != this->preconditions.end(); it++)
33         {
34             it->print();
35         }
36
37         std::cout << ", postconditions: ";
38         for(auto it = this->postconditions.begin(); it != this->postconditions.end(); it++)
39         {
40             it->print();
41         }
42
43     State engage(State s, GroundState ground) {
44         // Remove negated postconditions from state, add new facts to the state
45         for(auto it = this->postconditions.begin(); it != this->postconditions.end(); it++)
46         {
47             std::string name = it->name;
48             if(name[0] == '!') { // Remove this from the state
49                 name.erase(0, 1);
50
51                 auto position = s.end();
52                 for(auto stateIt = s.begin(); stateIt != s.end(); stateIt++)
53                     if(stateIt->name == name)
54                         position = stateIt;
55
56                 if(position != s.end())
57                     s.erase(position);
58
59             } else { // Addit to the state
60                 if(it->floating == false) { // Trivial case. Not floating? Add it directly
61
62                     s.push_back(*it);
63                 } else { // Otherwise, match it with the ground.
64                     // At this point, we know the name of the fact we want to create,
65                     // but we need to match up the arguments with the supplied ground
66                     state.
67                     std::vector<std::string> arguments = matchGrounds(it->params, ground);
68
69                     // Got all of this postcondition's params matched up to their ground
70                     // All that's left is to instantiate a matching Fact and add it to the
71                     state.
72                     s.push_back(Fact(name, arguments));
73
74                 }
75             }
76         }
77
78         return s;
79     };
80 };
81

```

codesamples/action.hpp

```

1 #pragma once
2
3 #include <iostream>
4 #include <vector>
5 #include <map>
6 #include <assert.h>
7
8 #include "goap/printable.hpp"
9

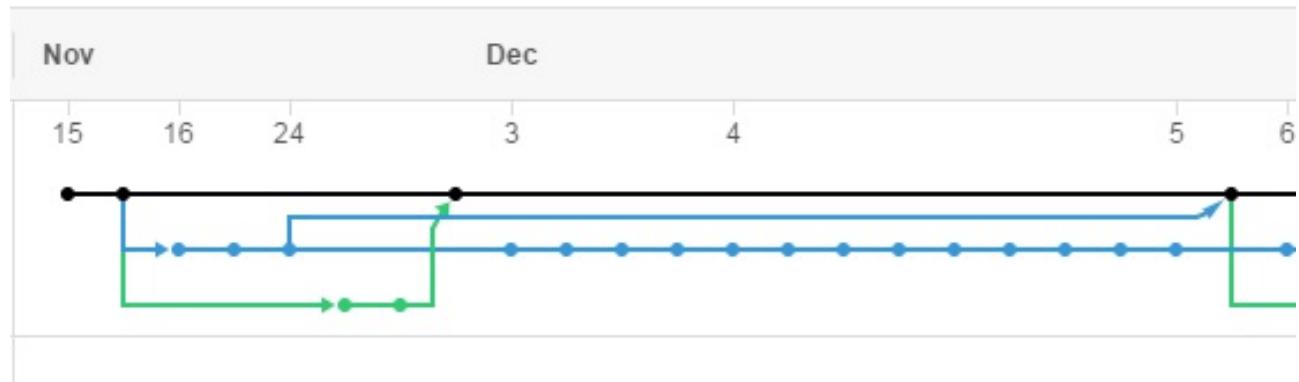
```

```

11  namespace Planner {
12      class Fact: public Printable {
13          public:
14              std::string name;
15              std::vector<std::string> params;
16              bool floating;
17
18          Fact(std::string _name, std::vector<std::string> _params, bool _floating=false):
19              name(_name), params(_params), floating(_floating) {};
20
21          void reify() {
22              this->floating = false;
23          }
24
25          void print() {
26              std::cout << this->name << "(";
27
28              for(auto it = this->params.begin(); it != this->params.end(); it++) {
29                  std::cout << *it << ", ";
30              }
31
32              std::cout << ")" (floating: " << (this->floating == 1 ? "true" : "false") << ")";
33          }
34
35      typedef Fact Step;
36      typedef std::vector<Fact> Goal;
37      typedef std::vector<Step> Plan;
38      typedef std::vector<Fact> State;
39      typedef std::map<std::string, std::string> GroundState;
40
41 // Match a list of parameters to their ground state.
42 std::vector<std::string> matchGrounds(std::vector<std::string> params, GroundState ground)
43 {
44     std::vector<std::string> arguments;
45
46     for(auto param = params.begin(); param != params.end(); param++) {
47         // Find this parameter in the ground state.
48         auto value = ground.find(*param);
49         if(value != ground.end()) {
50             arguments.push_back(value->second);
51         }
52     }
53
54     return arguments;
55 }
```

codesamples/fact.hpp

A.2.1



GitHub Branches

A.3 Literature Review

2015

Literature Review

GOAP

SAM MCKAY, ILLIAN GEORGIEV, AARON SWISS-HAMLET, JOSE MANUEL
DIEZ AND VLAD-EUGEN TANASE

Contents

Strips	2
Steering	3
Pathfinding.....	4
Map Generation.....	6

Strips

In order to fully implement a goal orientated action planning system we have researched heavily into the STRIPS methodology, this is one of the original methods of doing this and with slight updates can be utilised within our project.

Strips uses a regressive search algorithm to create an action plan, this works by have a desired state or goal and working backwards to fulfil the pre-requisites of the goal. For example, if the goal is to not be cold then a pre-requisite could be too be near a fire, in order to achieve that it would need wood, to get wood it has cut down a tree and to do that it needs to find an axe. So in this case its plan would be:

- Find axe
- Cut down tree
- Make fire
- Sit near fire

These goals can be in the form of a needs system i.e. hunger, energy, warmth, thirst and so on. These would each have a weight which would indicate their importance and the AI would prioritise those with a greater weight. This means that the search algorithm would need to recalculate when a step in its plan has changed in case the goal has changed, this however could make it more efficient in case it has discovered a quicker way of achieving the goal since it calculated the last plan.

Steering

Steering is useful for simplistic applications while still giving the impression of complex actions. The main implementation method involves vectors(math context). Steering model can be applied to collision avoidance, as well as following and tracking other entities and paths.

Steering has many different implementations. From flocking algorithms such as Boids to path tracking, collision avoidance, containment etc. Steering implements vectors to determine the direction in which an AI agent should move. These vectors are subject to change based on the implementation of steering and the current world state. A good implementation of steering is Anti Gravity movement and Vector Field movement. Both methods work in roughly the same manner. The AI agent has a direction/steer vector which it is trying to keep to.

Vector field/Flow field uses objects around the world and the goal position to create a grid of direction vectors which alter the original direction vector of the AI to have it avoid obstacles. This is useful for avoiding big and difficult to go through terrain.

Vector field method is somewhat limited by the resolution of the grid. An ideal solution is to have a good midpoint. Another possible issue with this method is moving objects.

While static terrain requires only one calculation in the case the grid resolution is high a big moving object that covers multiple grid squares will have to affect all the cells around its perimeter and even a bit beyond.

Similar to “Obstacle avoidance” described by Craig Reynolds, Anti Gravity works on roughly the same principle as Vector Fields however instead of a grid of vectors the AI agent tracks the distance between each object it can collide with and applies a steering vector opposite the direction of that object.

This approach is also similar to what is used in the Boids AI model developed by Craig Reynolds. Except in this case all other “entities” are static and only the Separation and Alignment rules are taken into account.

Path following allows for the use of predetermined paths in the game world. The agent still uses vectors to steer and implements the use of a future position. The future position usually will have an offset from the path this offset is used to correct the AI direction vector. While not 100% accurate to the path the AI will stay within a certain distance of it.

Steering can also be implemented in the context of moving entities. Following, intercepting, queuing and running away from. These behaviors work on the same principles as the ones listed above with the change that the entities/objects used as a reference for the calculations are not static.

Combining different steering implementations can give the AI agent multiple ways to navigate around the world based on information supplied to it by its sensors.

Pathfinding

Unlike steering method pathfinding can be used to solve complex problems such leave a maze and find the shortest distance between the start and end points (in case there are obstacles).

Pathfinding has several limitations and drawbacks:

It requires a comprehensive grid/graph with well defined connections between cells/nodes

Movement between those is not necessarily described by the algorithm used and usually has to be implemented separately.

Algorithms used for pathfinding are very resource intensive and can take a very long time to compute.

If a path is obstructed by a dynamic object the AI will not register that and possibly get stuck.

The two main approaches are directed and undirected pathfinding.

Undirected pathfinding is not widely implemented and it uses one of two main algorithms, Breadth-first and Depth-first. They both search through the connected nodes until they find a solution with no regard as to the difficulty, time it might take to get to the final goal and while similar the two algorithms work in almost the opposite way.

Breadth-first expands all nodes, at one level, revealing all the nodes they are connected to. Only when all nodes from the current level are expanded will it move down to the next.

This is useful where the node hierarchy has multiple solutions at different depth levels. Breadth-first will find the solution that takes the least steps(node expansions).

As the name suggests Depth-first does not focus on expanding all nodes originating from a single node before moving on. It looks at the children nodes until it reaches a node which has no new ones originating from it. This method is much more efficient than Breadth-first in situations where you have a very uniformly shaped tree of nodes with a lot of solutions.

The directed pathfinding is more common in games since it provides the shortest/fastest route to a target location. However it is much more time consuming to compute than the undirected methods.

Directed algorithms work on a cost system described by the developer. Costs or weights can be anything from the physical distance between nodes in the game, to estimated time of completion or even arbitrary values put in by the programmers/designers to “guide” the algorithm in a certain direction.

Like with undirected pathfinding there are two main strategies to finding the shortest: uniform search and heuristic search.

Uniform search gives an optimal result picking the easiest route possible every time however it is very inefficient and may take a very long time to complete.

As the name suggests heuristic search uses a heuristic to determine the route. This is the much more efficient method however but it does not search through all possible paths and not always is the final solution the most optimal.

The two most used algorithms for directed pathfinding are Dijkstra(uniform search) and A*(combines uniform and heuristic search). Other algorithms exist which are mostly variations of the ones already mentioned.

These variations or evolutions are developed in order to solve some of the issues that are present whenever pathfinding is involved. A* itself is an evolution of Dijkstra.

PRA* is an evolution of the A* algorithm. It speeds up the A* algorithm by first generating a solution for a more abstract version of the search space. This way the A* algorithm has to look through a fraction of the original node/grid mesh.

D*(dynamic A*)this algorithm takes into account changes in the cost of a path after it has been calculated and the AI has been set in motion. This solves the problem with dynamic objects but adds a lot more strain on the CPU in order to accomplish this.

Variations of the A* algorithm go on with each variation improving the original in some way. Here are some of them and how they are influencing the original they are built upon.

Dijkstra – finds all possible paths through a graph/grid.

A* - Uses a heuristic to find the shortest/most optimal path.

IDA* - Iterative Deepening A*

This version tries to speed up A* starting with an approximate answer. With each iteration it expands more and more nodes and tries to match the overall weight/cost to the initial estimate. Somewhat based on the Depth Search algorithm for looking through trees.

D*/LPA* - Dynamic A*/ Lifelong Planning A*

These versions solve the problem of dynamic environments in exchange for requiring a lot of space. These versions keep all details of the last computed path and match it to the current graph/grid state to determine if a recomputed is needed.

PRA* - Partial Refinement A*

Creates a “lower resolution” version of the original graph/grid thus speeding up the algorithm.

LRTA* - Learning in Real-Time A*

substantially lowers the first move delay present in standard A*. It accomplishes this by first doing a very small scale calculation to determine the first step and then it finds paths to the end goal which may be suboptimal. Finding an optimal path is much more expensive compared to A*

Pathfinding gives the best solutions however it can be very resource intensive based on the implementation. Current algorithms and their variants try and balance speed, resource requirement and quality of result. A specific implementation can be chosen based on the needs of the AI agent.

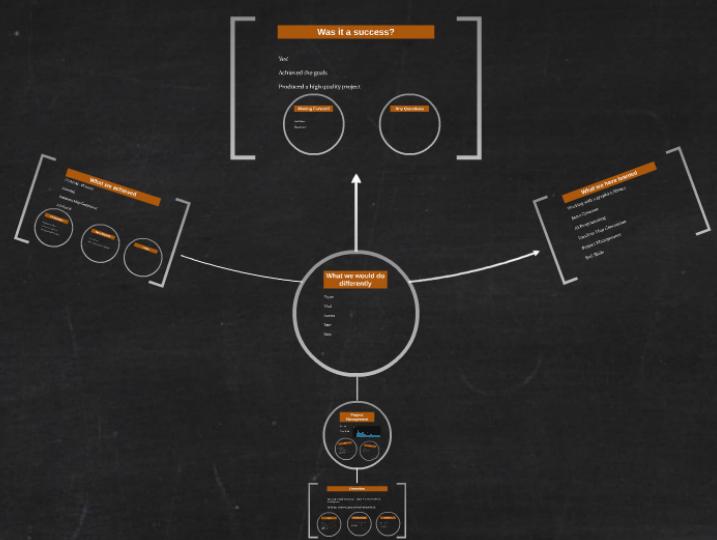
Map Generation

Research conducted into random map generation for 2D games.

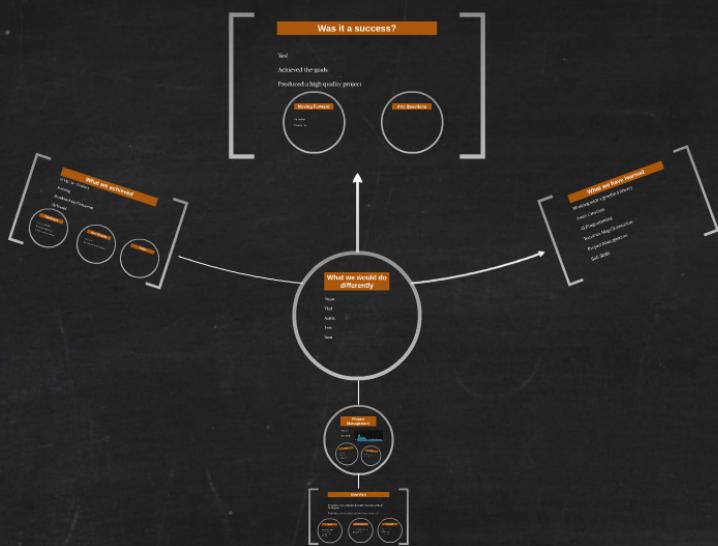
Easiest approach is to create tiles, however other options are Height map, Perlin Noise but those are more complicated than what we require for the project and as such we shall be sticking with the tiled approach. This will allow to create plenty of different maps in which to test our AI, we can also utilise the random generator for the creation of objects throughout the scene allowing us to spend more time upon the action planning section.

The maps will need to be exported into an XML format that can be read with a TMX reader that can be used with SFML, the advantage of having them in XML format is that we can also edit them manually should we need to.

A.4 Final Presentation Slides



Goal Orientated Action Planning



Goal Orientated Action Planning

Overview

To create a Goal Orientated Action Planning Artificial Intelligence

To design a survival game in which to test the AI

Goals

- Be able to generate a plan
- Carry out the plan
- Visualize this in the 2D environment

Overall Outcome

- Created an AI capable of planning
- Demonstrated it in a 2D Environment

Teamwork

- Success as a team
- Skill development
- Communication

Goals

Be able to generate a plan

Carry out the plan

Visualize this in the 2D
environment

Overall Outcome

Created an AI capable of planning

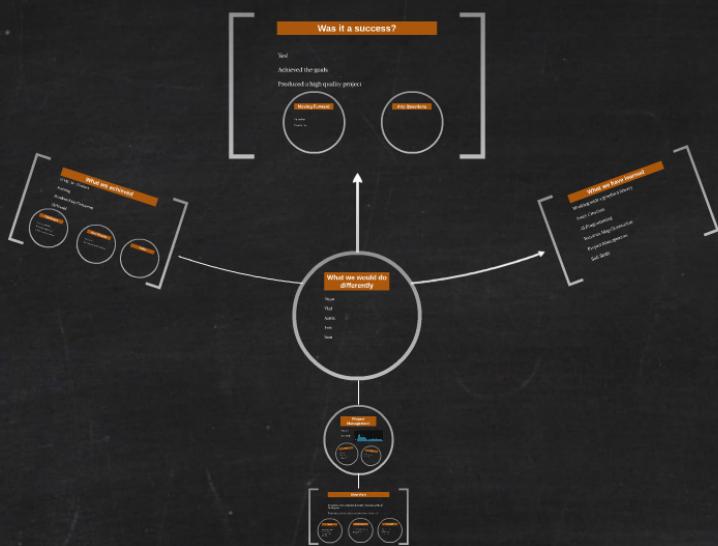
Demonstrated it in a 2D
Environment

Teamwork

Success as a team

Skill development

Communication



Goal Orientated Action Planning

What we have learned

Working with a graphics library

Asset Creation

AI Programming

Random Map Generation

Project Management

Soft Skills

What we achieved

GOAP AI - Planner

Steering

Random Map Generator

2D World

Challenges

- Keeping to deadlines
- Pacing and staying power
- Working in a multidisciplinary

New Wizards

- Great Success
- Interest from potential employers

Video

Challenges

Keeping to deadlines

Pacing and staying power

Working in a multidisciplinary

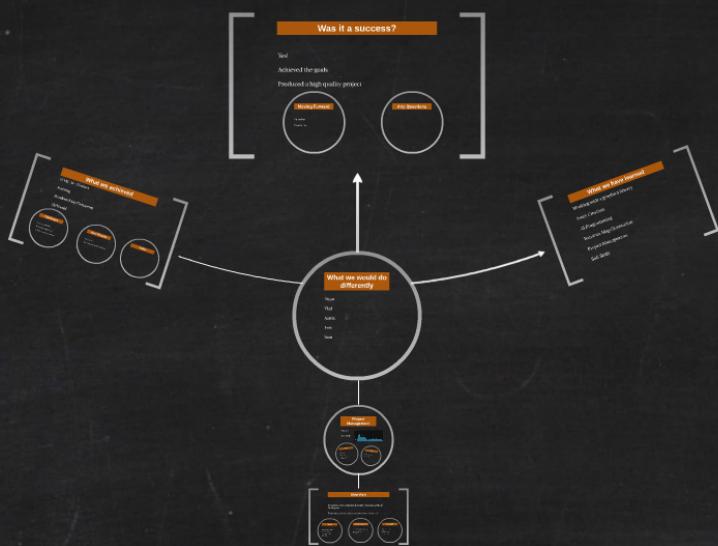
New Wizards

Great Success

Interest from potential employers



Video



Goal Orientated Action Planning

Project Management

Axosoft

New Skills



Pro

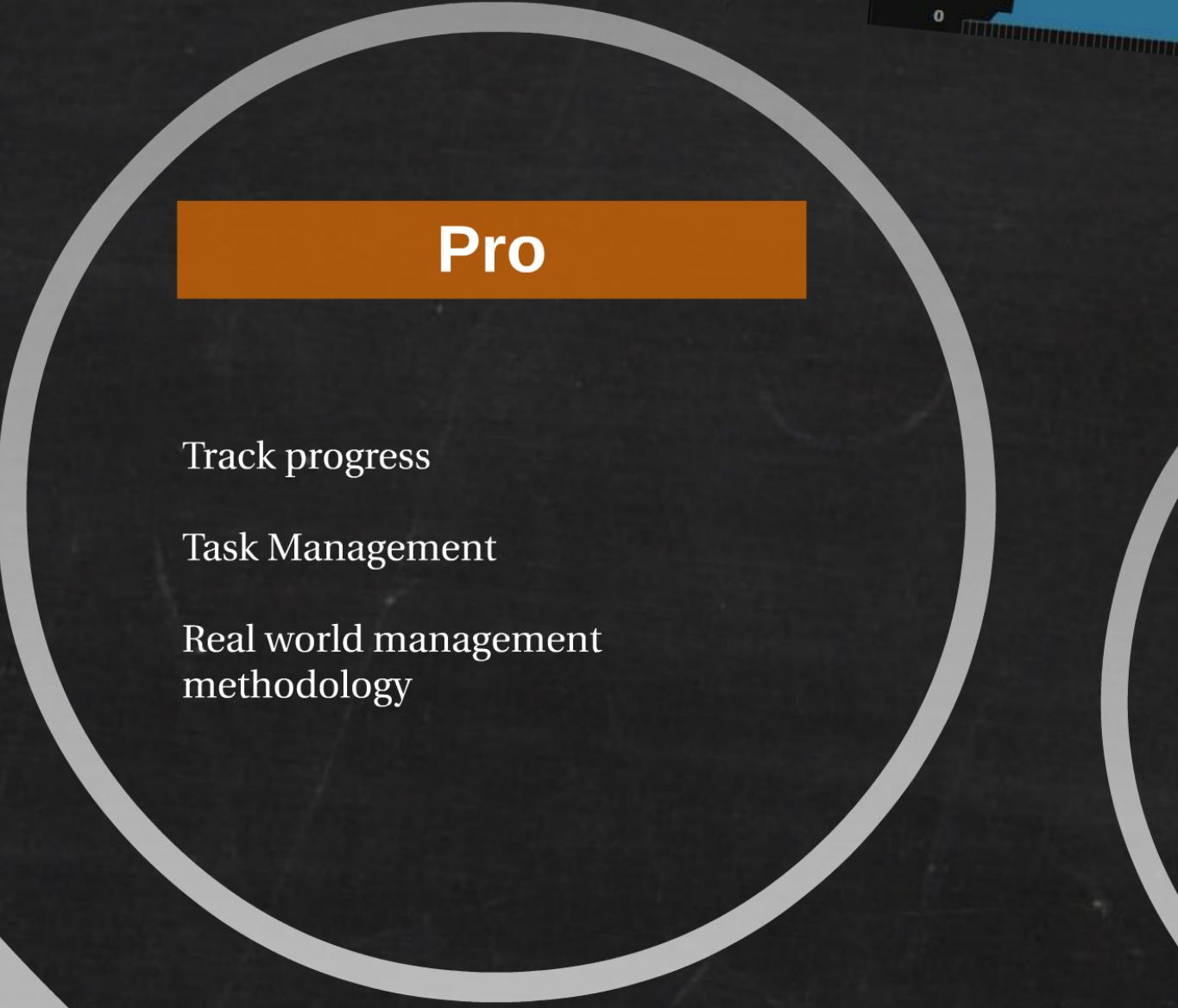
- Track progress
- Task Management
- Real world management methodology

Con

- Not always up to date
- Learning curve

Project Burndown





Pro

Track progress

Task Management

Real world management
methodology

Con

Not always up to date

Learning curve

What we would do differently

Iliyan

Vlad

Aaron

Josè

Sam

Was it a success?

Yes!

Achieved the goals

Produced a high quality project

Moving Forward

Portfolios
Experience

Any Questions



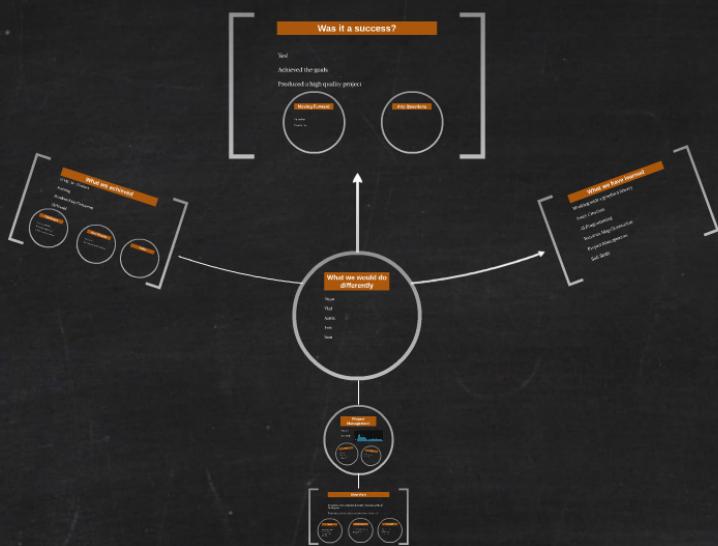
Moving Forward

Portfolios

Experience



Any Questions



Goal Orientated Action Planning