**SCHOOL OF COMPUTING**

**UNIVERSITY OF TEESSIDE**

**MIDDLESBROUGH**

**TS1 3BA**

**AI for Games   
(COM3049)**

**ICA3 Group Report**

**Jackdaw**

**Jarred Davies**

**Lucy Keel**

**Daniel Quirke**

Table of Contents

[1 System Architecture Overview 3](#_Toc451893219)

[2 Motion planning 4](#_Toc451893220)

[3 Task planning 5](#_Toc451893221)

[4 Plan Executive 5](#_Toc451893222)

[5 Results 7](#_Toc451893223)

# System Architecture Overview

System Architecture

Unity3D was chosen for the visualisation component so that we could focus on the rest of the task at hand. The core controller is the executive. It is responsible for communicating with the planners in order to generate action sequences or to move entities around the map. It communicates with the motion planner by telling it what to move and where and gives updates on the world state so that new paths can be planned if needed. The motion planner in return, will tell the executive new updates to the world, based on entity positions changing. The motion planner stores the path nodes for each of the moving entities.

The executive also communicates with the task planner. This is important for generating plans to complete the global objective. Actions are also verified using the task planner before they are committed and the task planner will always return a sequence of actions to complete a goal.

Finally, the executive commits to actions and directly affects the visualisation of the map by spawning buildings, changing resources and more. It stores the current world state at all times along with the global objectives and the actions to complete the current objective.

# Motion planning

For the motion planning part of our AI, we decided upon implementing a standard A\* search algorithm. A\* was selected for a number of reasons. Firstly, it can guarantee an optimal path as long as we use an admissible heuristic and a path from the starting node to the goal exists. Secondly, its performance was deemed acceptable for the small maps we were using.

In the design of the algorithm a couple of considerations where made for performance. As A\* cannot return a partial solution, it must traverse the graph until it has found the goal. There was a possibility that for particularly challenging paths A\* could stall the entire simulation until it has found a solution. To counteract this problem, we devised two improvements to make stalling unlikely. Firstly, as a majority of the paths required are going to be between buildings and other common points, all solutions between requested nodes are stored in a list. If the same path is requested again it is simply retrieved from the list to save on processing time.

Secondly, it was decided that the path planning must be multithreaded. This meant that when an agent requests a path from the algorithm, it returns a Task object. This task object contains a Boolean that indicates whether the path has been solved. The task object is then checked by the agent every tick to see if a solution has been found. This meant that if a particularly complex path is needed, it will only effect the performance of that individual agent and not the rest of the simulation.

For the evaluation heuristic, it was decided that a heuristic based on Chebyshev distance would be used. We decided to use Chebyshev as it is admissible for maps that include diagonal movement, unlike the Manhattan distance heuristic which is only admissible for maps without diagonal movement.

To allow multiple heuristics to be used easily by the path planner, an interface was defined for the heuristics. This meant that we could switch between differing heuristics easily and on the fly. The major benefit of this was the flexibly it afforded us. We could change the heuristic depending on the needs of the simulation. For instance, if we found that path planning was too expensive, we could relax the bounds of the algorithm by using a weighted heuristic. This would allow us to make the algorithm less expensive in exchange for the paths optimality being worse.

# Task planning

(1000 words max and **diagrams**)

# Plan Executive

It was discovered early on that the best tactic to win in this game was to kill the opposing teacher. This is because without a teacher, people cannot be trained which then puts the other player in the driving seat as they can proceed to killing off the rest of the opponent to be crowned champion.

Because of this, the plan executive was designed around this tactic. The first milestone step to achieving this is to build a barracks. This allows riflemen to be trained by the teacher which in turn will allow for hunting down and killing the opposing teacher.

The executive stores the world state which is obtained from the game manager. The world state would be passed into the task planner in order to generate plans if the interface to the task planner was implemented. The generated plan would then be stored as a first in first out queue of actions and each action can then be executed in sequence.

Before executing an action, the task planner would be queried to verify that, with the current world state, the action is still possible. This would stop an entity trying to attack another entity that died 10 ticks ago and a lot more situations like this.

If the action is still valid, it can be executed. Unity Scriptable Objects would be best suited for this because all of the actions have the same basic functions. They all have a time to complete, a requirement and an end product. This would have made actions very easy to interact with however we do not have this infrastructure in place.

If an action is to move a troop, the motion planner will be called and given the current world state. This is so that it can generate the path and start to move the troop towards the goal. When a change in the world state is detected, the motion planner is updated with the new world state so that it can generate a new path based on the new world state.

Executive Flow

The approach for the executive was a partial plan, partial execution. It knows that the global objective is to kill the opposing teacher but rather than generating a full path from the start, where lots of changes in the map are bound to be made, it breaks the problem into sub problems. As explained before, the first step is to build a barrack, then to train some riflemen and finally to kill the opposing teacher.

For the partial execution, only 1 action is processed at a time and it would be checked for validity before doing so. If it is not valid then the task planner is queried to generate a new set of actions that will allow the current action to be completed. This may be because there is now a building in the position that you wanted a building. The new steps are then added to the front of the queue and completed as normal.

# Results

Present some characteristic examples (a few) of what you have achieved (include a variety of examples, rather than several of the same type).

Make good use of snapshots in order to illustrate your results.