Technical University of Denmark

02285 Artificial Intelligence and Multi-agent Systems

Mandatory Assignment 3

Authors:

Andreas Hallberg Kjeldsen
s092638@student.dtu.dk

Morten Chabert Eskesen
s133304@student.dtu.dk

Peter Carlslund
s113998@student.dtu.dk

June 2, 2014

1 Introduction

Andreas & Morten

This project was produced while taking the course Artificial Intelligence and Multi-agent systems at the Technical University of Denmark. The project had to be done in groups of 3 to 6 people. For the development of the project, there were 3 iterations, this is the report detailing the third and final iteration. Each section and subsection is annotated with whom contributed to the writing of the section. Sections regarding implementation specific areas are written by the persons who wrote the actual implementation.

1.1 Scenario

Andreas & Morten

There are 28 agents with different roles. They are dropped into an unknown environment. The agents have properties (health, max health, energy, max energy), a visibility range and a role defining which specific actions they can do. The environment is a graph consisting of vertices and edges, where vertices have a certain value and edges cost a certain amount of energy to traverse. Lastly there is the notion of a score which is the accumulated sum of the zone score for a team during the game.

1.2 Problem Analysis

Andreas & Morten

The scenario of the game introduces a set of problems to overcome.

1.2.1 Gathering Knowledge

Andreas & Morten

The agents have no prior knowledge of the environment, no knowledge about where the other agents are in relation to each other, nor where the opponent agents are. The agents must be able to share their knowledge and keep track of their opponents.

1.2.2 Working Together

Andreas & Morten

The agents might wish to perform the exact same action, this is not beneficial, therefore the agents should be able to work out whom should do what. Preferably in such a way that the agent able to perform the action with the probable best outcome, is chosen to perform it.

1.2.3 Scoring

Andreas & Morten

The agents must coordinate their actions in accordance with an overall strategy that works towards achieving a good score.

1.2.4 Agent Strategy

Andreas & Morten

A strategy for the agents must be defined. How aggressive should the Saboteur agent be. When should the agents request to be repaired by the Repairer agent. For how long

should the agents focus on gathering knowledge and when should they focus on achieving a good score.

2 Environment

Andreas

The environment is randomly generated, that means assumptions about the environment should not be made. Further the agents are placed at random. The agents are allowed to communicate and share their knowledge.

2.1 Map

Andreas

The map of the environment is being represented as an edge weighted graph. Each vertex has a value indicating its score, each edge has a weight indicating the energy cost of traversing the edge. The agents cannot determine specific coordinates of the vertices, thereby making it hard to determine the direction an agent would go.

The distance between two connected vertices will be referred to as a *step*. The amount of steps away the agents can perceive, varies from 1 to 3. Whenever an agent perceives a vertex, the agent will remember the vertex, including the connecting vertices and the edges between them.

The vertices has to be probed to obtain information about how valuable they are. Only the *Explorer* agent can probe vertices. If a vertex is not probed, the value of the vertex is set to be 1. The edges has to be surveyed to obtain information about how much it costs to traverse them. All agents can survey edges. The costs of traversing an edge is not depending on whether the edge is surveyed or not. Knowing the edge costs gives advantages when finding paths for the agents to follow, i.e. shorter paths with lower costs.

2.2 Knowledge

Andreas

The agents share all their knowledge, that is, they have a centralized knowledge base. The agents therefore also have the same perception of the environment. The agents share all their new percepts before planning what they should do next. Having a centralized knowledge base, eliminates the need for communicating messages regarding perceptions of the environment, i.e. new vertices, new edges, opponent spotted.

3 Strategy

Morten

At game start the agents have no knowledge about the environment. They do not know where their fellow agents are located nor do they know where the opponent agents are. Therefore in order to be able to strategize properly for the game, the games is split into two phases. The two phases have been named the *Mapping* and *Zone Control Mode*.

Andreas

3.1 Mapping

When the game starts, the first thing the agents focus on is mapping the environment. The Explorer agent focus on probing vertices, looking for unprobed vertices etc. The other agents walk around randomly looking for unsurveyed edges and opponents. The agents will only try to survey the surrounding edges if a specific amount of unsurveyed edges are visible. This avoids spending too much time surveying when only a few unsurveyed edges are near.

Andreas & Morten

3.2 Zone Control Mode

At step 150 in the game, Zone Control mode is activated. This is based on the assumption that a great part of the environment is mapped at step 150. Zone Control mode is the phase where the agents defend a zone of the map, i.e. a subgraph of the entire graph. In Zone Control mode the agents only defend the zone and do not care about what the opponents are doing unless the opponents are attacking the agents or threatening the zone in any way.

Zone Control mode is activated because the goal of the game is to achieve the highest score. The game grants points based on the zone score for the team. We have several algorithms that finds a high scoring zone in the graph, which we will cover in the section Zone Control. At each 150th step while in zone control mode, a zone recalculation is made, to see if a higher scoring zone can be found or if there is a way to expand the current zone.

3.2.1 Path Planning

During Zone Control mode the agents are to place themselves at specific vertices. Each agent must plan their way to their specific vertex. We have chosen to use Dijkstra's Algorithm for path planning. The path chosen is based on the cost of the edge and the amount of steps required to reach the specified vertex. We prefer paths with lower edge cost but also a path that requires fewer steps. A path could contain a small amount of edges to traverse but they could all be costly, therefore it might be advantageous to take a longer path with lower edge costs to avoid spending too much time recharging.

Morten

3.3 Reflection

Our strategy for the game is quite simple. When Zone Control mode gets activated the agents are practically standing still with the hope that this would outscore the opponent team. This might however not be the case for several reasons.

- If the agents haven't mapped, and probed, the highest scoring zone of the map and the opponent controls this zone. There is no plan in place to counter this scenario and will therefore be outscored by the opponent and loose.

- If the Explorer agents in the first phase haven't probed many nodes due to being disabled then the agents may end up defending a zone that does not achieve a high score. This again is a scenario that there is no plan in place to counter as the Explorer agents will not probe any nodes not in the controlled zone when entering zone control mode.

There are other scenarios to consider but what one ultimately finds is that our strategy is quite naive. Also if the agents try to defend a zone where the opponents are, the agents will fight for this zone until a zone recalculation is made to find another zone. The strategy would therefore be an obvious thing to improve if more time had allowed it, as in general the agents do not counter their opponents as much as they should.

4 Agents

Andreas

Each agent has a specific role. Depending on their assigned role, certain properties and abilities are available to the agent. The available agent roles are Explorer, Inspector, Repairer, Saboteur and Sentinel.

The agents make use of the *Belief-Desire-Intention* software model. This means that each agent has its own beliefs, its own desires and its own intentions. A belief is an assumption about the environment, though the agents are certain that a vertex will not change position, they are not certain that an opponent agent spotted will not move away. Therefore not all beliefs cannot be determined to be facts. A desire is something the agent desires to do. The role specific actions for the agents, determine their desires. The Explorer agent desires to probe unprobed vertices, the Saboteur agent desires to sabotage the opponent agent and so forth. The intentions of the agents are immediate intentions like, "I wish to go to vertex v233, on my I will have to pass through vertices v182 and v377".

An agent can be disabled, meaning it has 0 health left. A disabled agent is still able to `Goto` a vertex, `Recharge` to get energy and `Skip` its turn. A disabled Repairer agent can also use `Repair`. A disabled agent does not contribute to the zone scoring.

Each action available for the agents requires a certain amount of energy. The amount of energy required depends on the action. If an agent tries to perform an action that requires more energy than the agent has available, the action will fail. An agent can get more energy by using the `Recharge` action. If an agent is disabled it can still `Recharge` but the amount of energy recovered is less than if the agent wasn't disabled.

4.1 Agent Base

Andreas

The agents have some properties, functionality and actions in common. The agents start by interpreting their percepts, thereby gathering new knowledge and intel regarding the opponent team. The agents are able to determine if an agent is nearby based on the

shared knowledge of the environment and the opponent positions. The agents are able to find a path from their current vertex to a goal vertex. The agents are able to make distress calls, which indicates that they're in need of repairing.

In the event that an agent tries to perform an action but receives notification that they do not have enough energy to do so, the agent is forced to perform a `Recharge` before attempting to do the same action again. This avoids the case when an agent has an intention to do something, but not enough energy to do it, which could lead to an agent being stuck.

When Zone Control mode is activated, the agents will be given a goal vertex to `Goto`. The agents will make a an individual plan for reaching the goal vertex.

4.2 Explorer Agent

Andreas

The Explorer agent is the only agent capable of probing vertices. The Explorer agent is able to perform the following actions: `Skip`, `Goto`, `Recharge`, `Survey` and `Probe`.

Phase 1: Mapping

The Explorer agent is a valued agent during the first phase. The Explorer should probe as many unprobed vertices as possible. Due to this, it prefers probing over surveying. If the Explorer is standing on an already probed vertex, it will find the closest unprobed vertex, `Goto` it and `Probe` it. The Explorer has a visibility range of 2, which makes it capable of perceiving a good amount of vertices and edges while walking around the environment. The Explorer will try to survey unsurveyed edges if it can see at least 6 unsurveyed edges. The Explorer agent is vital for achieving a good zone score, as unprobed vertices only grants 1 point. Therefore the Explorer agent makes a distress call if its health drops below 40%. Whenever the Explorer energy drops below $\frac{1}{3}$ it will `Recharge`.

Phase 2: Zone Control Mode

When Zone Control mode is activated, the Explorer will `Goto` a goal node to maintain the zone. While maintaining a zone, the Explorer will stand still, performing `Recharge` if energy is not full otherwise it will `Skip`. If an unprobed vertex is found within the maintained zone, the Explorer will `Goto` it and `Probe` it to further increase the zone score. If the Explorer becomes disabled, it will make a distress call.

4.3 Inspector Agent

Andreas

The Inspector agent is the only agent capable of inspecting opponents agents. Inspecting an opponent agent reveals the agents role and properties, i.e. health and energy. The Inspector agent is able to perform the following actions: `Skip`, `Goto`, `Recharge`, `Survey` and `Inspect`.

Phase 1: Mapping

During the first phase the Inspector agent will help with the mapping by surveying unsurveyed edges. The Inspector has a limited visibility range of 1 and can therefore not see very far. Due to this, the Inspector can only `Survey` a few edges at a time. The visibility range of the Inspector also limits its ability to spot opponent agents. The Inspector agent will try to `Inspect` an opponent when its within the Inspector visibility range. An opponent will only be inspected once. If no unsurveyed edges have been found and no previously inspected opponent is nearby, the Inspector will `Goto` a random vertex. If the energy of the Inspector drops below 3, it will `Recharge`. If the health of the Inspector drops below 40% it will make a distress call.

Phase 2: Zone Control Mode

During Zone Control mode, the Inspector will `Goto` a goal vertex and stand still. If an opponent is visible and not previously inspected, the Inspector will `Inspect` it. As in phase 1, if the Inspectors energy drops below 3, it will `Recharge`. While standing still and fully recharged, the Inspector will simply `Skip` its turn. If the Inspector becomes disabled, it will make a distress call.

4.4 Repairer Agent

`Andreas & Peter`

The Repairer agent is the only agent capable of repairing other agents. The Repairer agent is able to perform the following actions: `Skip`, `Goto`, `Parry`, `Recharge`, `Survey` and `Repair`.

Phase 1: Mapping

When the mapping process is going on, the Repairer agent will help with the mapping by surveying unsurveyed edges. The Repairer has a limited visibility range of 1 and can therefore not see very far. The most important job for the Repairer is to help its fellow agents when they're in distress. The Repairer agent responds to distress calls. When responding to a distress call, the Repairer will `Goto` the fellow agent in distress and `Repair` them. If no agents are distressed, the Repairer agent will walk around randomly. If the energy of the Repairer drops below 3, it will `Recharge`. The Repairer will not make a distress call if its disabled, this is because the Repairer can still use `Repair` even when disabled. The Repairer agent cannot repair it self.

Phase 2: Zone Control Mode

When in Zone Control mode, the Repairer will `Goto` a goal vertex and stand still. If a fellow agent is making a distress call the Repairer agent will respond to it. After the Repairer has helped a distressed fellow agent, the Repairer will return to the goal vertex it was standing on before helping the fellow agent. As in phase 1, if the Repairers energy

drops below 3, it will `Recharge`. While standing still and not disabled with no distress calls to handle, the Repairer will `Parry`.

4.5 Saboteur Agent

Andreas & Peter

The Saboteur agent is the only agent capable of sabotaging the opponent agents. The Saboteur agent is able to perform the following actions: `Skip`, `Goto`, `Parry`, `Recharge`, `Survey` and `Attack`.

Phase 1: Mapping

In the first phase, the Saboteur will help with the mapping by surveying unsurveyed edges. The Saboteur has a limited visibility range of 1 and can therefore not see very far. The most important job for the Saboteur is to attack the opponent agents, hopefully disabling them. The Saboteur attacks an opponent agent when it sees one. To avoid getting stuck by continuously attacking the same agent over and over, the Saboteur will attack the same opponent a maximum of 4 times in a row. The Saboteur does not evaluate the result of its attack, meaning a parried attack will not affect the behavior of the Saboteur. If the energy of the Saboteur drops below 3, it will `Recharge`. If the health of the Saboteur drops below 40% it will make a distress call.

Phase 2: Zone Control Mode

In Zone Control mode, the Saboteur will `Goto` a goal vertex and stand still. If an opponent agent steps into the zone where the Saboteur is, the Saboteur will attack the opponent. A maximum of 4 attacks in a row is still in applied. As in phase 1, if the Saboteurs energy drops below 3, it will `Recharge`. While standing still and not disabled, the Saboteur will `Parry`. If the Saboteur becomes disabled, it will make a distress call.

4.6 Sentinel Agent

Morten

The job of the Sentinel agent depends on phase of the game and whether or not the agent is disabled. The Sentinel agent is able to perform the following actions: `Skip`, `Goto`, `Parry`, `Recharge` and `Survey`.

Phase 1: Mapping

In the first phase the Sentinel will help with the mapping by surveying unsurveyed edges. The Sentinel has a visibility range of 3, meaning it can survey a lot of edges at once. If there is nothing to survey within its visibility range, the Sentinel will walk around randomly to further help the mapping process.

Being able to survive by itself is important for the Sentinel agent because we value it as one of the lesser valued agents. Therefore the Sentinel agent should burden the Repairer agents as little as possible. That is why the Sentinel agent will `Parry` or run

away from an opponent if an opponent is close during the Mapping phase. The Sentinel agent will `Parry` a maximum of 5 times in a row, as we wanted to avoid the agent being stuck at some position. If the Sentinel has less than $\frac{1}{4}$ of energy then it will `Recharge`. If the Sentinel is disabled it will make a distress call.

Phase 2: Zone Control Mode

In the second phase the planning for the Sentinel agent becomes a lot more simple. The Sentinel will still `Recharge` if it has less than $\frac{1}{4}$ energy left and `Goto` a goal vertex if it receives such a goal vertex. Otherwise the Sentinel will parry if an opponent is close to defend the zone and if no opponent is close it will `Recharge`. Note that in this phase there is no maximum amount of times the Sentinel can `Parry` it will `Parry` until its energy is less than $\frac{1}{4}$ of its total. As in phase 1 the Sentinel will make a distress call if it is disabled.

5 Planning

Andreas

Having multiple agents all doing what they think is the best thing to do, is not always the most optimal way to go about it. Therefore we wanted to make sure that the agents coordinate their actions thereby working together to achieve an overall good result. We also wanted to leave no agent behind, hence we made it possible for the agents to ask for help.

5.1 Planning Center

Andreas

We created a system, we'll be referring to as the *Planning Center*. The Planning Center keeps track of the actions the agents intent to perform. It refuses actions that would be a duplicate of an already planned action, while also preferring actions that are more beneficial than already planned actions. A more beneficial plan could be going to an unprobed vertex in as few steps as possible. The agents will only perform their action, when all agents have an action to perform that does not conflict with the others.

The agents plan their turn in a sequential order. The Planning Center therefore acts as a bidding system. The agents bid on the action they wish to perform, using how much the action would cost as their offer. Some actions are 'first come, first serve', these include probing of a specific vertex, surveying from a specific vertex, attacking a specific opponent agent or repairing a specific fellow agent. Other actions are given to the agent with the best offer. Best offer is determined based on the action. For going to and probing an unprobed vertex, the best offer is the lowest amount of steps required to reach the vertex. If an agent is outbid, the agent will have to come up with a new action to perform.

5.2 Distress Center

Andreas

When an agent gets attacked it might get disabled. To remedy this, the agent can be repaired by a Repairer agent. We created a *Distress Center* in order for the Repairer agents to know which other agents are in need of help. When an agent is in need of help, we say that the agent is distressed. A distressed agent will make a distress call. The distress call gets logged in the Distress Center. The Repairer agents can respond to the distress calls, by asking the Distress Center about possible distressed agents. The Repairer agent closes to the distressed agent will be assigned to help. In the event of more distressed agents than Repairer agents, the distressed agents will be helped in "closest-agent-first-order". If only a few agents are distressed, only the minimum required amount of Repairers will respond to the distress calls.

6 Zone Control

Morten

Zone control is where the agents defend a zone from their opponents. As mentioned earlier a good zone is a high scoring zone. In this section we will outline and compare the different algorithms we have implemented to find a high scoring zone, where only one is used in the final implementation. Finally we will explain how we guard the zone.

6.1 Algorithms

Andreas & Morten

There are a lot of approaches to achieving a good zone. One we have discussed a lot was finding a corner and then isolating this corner by making a "frontier" in the map. This will cause a lot of vertices to be dominated by us and then it will probably achieve a high score, even though the decision to isolate the corner had nothing to do with calculating the score of the vertices. However we have not found a way to identify corners which has lead us to consider other approaches. In the following sections we will describe these approaches and the algorithms we have implemented.

6.1.1 Isolated Subgraph

Morten

The Isolated Subgraph is an approach where we pick a high valued vertex and then build a zone around it. The vertex becomes the center of the zone. The algorithm depends on how many agents we can use to build a zone. It will expand from the center vertex until we reach a number of vertices higher than the amount of agents. Then it will place the agents such that we dominate all the nodes inside this zone.

6.1.2 Max Sum Component

Andreas

The Max Sum Component works by splitting the explored environment up into smaller connected components. A connected component is a subgraph. The connected components are found by iterating over all the vertices. For each vertex we add it to a set of already seen vertices and then we iterate over the vertices connected to the current

vertex. If a vertex has already been seen it is skipped. At some point all connecting vertices have been visited and we have found a component. If more vertices are to be iterated a new component is made and the process is repeated. Finding the connected components in the environment takes linear time.

A restriction on the components to find has been set, that is, the components must be no larger than the amount of agents available. To decrease the size of the components found, the vertex with lowest value is removed from each component. At last a set of components of maximum allowed size remains. The value of a component is the sum of all values of the vertices. The component with maximum sum is chosen as the zone to control.

6.1.3 Simulated Annealing

Morten

Simulated Annealing is an approach which locates a good approximation to the global optimum for a given function in a large search space. Simulated Annealing is a good approach for locating a high scoring zone because the agents only have a certain amount of time to respond to the server and the map is a large search space. With other words the goal is to find an acceptably good solution in a fixed amount of time, rather than the best possible solution. The smart thing about simulated annealing is that it will probabilistically choose a solution if it is worse than the current solution found and always choose the new solution if it is better. The reason that it is smart to accept a worse solution is that it will never get stuck at a local optimum.

Our implementation of Simulated Annealing uses the previous algorithm described which finds a connect component of a fixed size and then expands this component for the purpose of getting a higher scoring zone. We discovered that the algorithm Max Sum Component was really fast so instead of choosing a random vertex and try to expand around that, we could just as well use this algorithm. The algorithm expands the zone by using the rules described for controlling a zone. It picks a random vertex which is not already dominated by us and is adjacent to one of the vertices we already have an agent placed at. The algorithm runs until $temperature < 1$, where $temperature$ is 1000 initially and each iteration $temperature$ will "cool" by 2.5%. The function to deem a solution acceptable (if the new solution is not a higher scoring zone) is Euler's number e raised to $(currentScore - newScore)/temperature$, where $currentScore$ it the score of the current created zone and $newScore$ is the score of the newly created zone.

$$e^{\frac{currentScore - newScore}{temperature}}$$

If this function is greater than a (pseudo) random number between 0 and 1 we accept the new solution.

6.1.4 Comparison

Andreas & Morten

A comparison of the algorithms would give an indication of their performance during a game. We chose 25 different seeds for the map generation. For each algorithm, a game using all of the different seeds was played. The comparison was focused only on achieving a good score, hence the opponent team was limited to a single agent, that did nothing but `Skip`.

Algorithm	Worst Score	Best Score	Average Score
Isolated Subgraph	79916	517131	207911
Max Sum Component	157072	222213	179179
Simulated Annealing	165072	216953	186135

6.1.5 Our Choice

Andreas & Morten

Based on the above comparison, we were intrigued to use the Isolated Subgraph algorithm as it had the best score and best average score. A big downside to using the Isolated Subgraph, is that it scores varies a lot. As the maps are randomly generated, there is no way of telling if we're lucky enough to get a good score or end up with a low score.

Both Max Sum Component and Simulated Annealing are steady. Simulated Annealing has the highest worst score and and a better average score than Max Sum Component. Simulated Annealing is also better at adapting to the opponent team strategy, where Max Sum Component would often find the same zone. Using the same zone for the whole game could give the opponent team some advantages.

We also ran games using the different algorithms while competing against a full opponent team. Both Isolated Subgraph and Max Sum Component had some serious issues with the opponent intruding the zone, Simulated Annealing adapted better to the situation. We therefore chose to use Simulated Annealing in our solution.

6.2 Guarding The Zone

Morten

The agents guard their zone by using `Parry` if possible. This increases their chances of surviving if an opponent attacks. If opponents are able to intrude the zone, the agents attack them with the hope of killing them and regaining control of the zone. The agents also guard the zone by repairing the agents making distress calls due to being disabled.

7 Flaws

Peter

Below is a list of flaws we have identified in our solution. We believe most of these flaws could have been fixed, if we have had more time.

7.1 Generating A Proper Map

Peter

When discovering the vertices and edges in the first phase of the game some sort of a coordinate system could have been made. Using the vertices and the paths between them, it could be possible to make a relation between the vertices and thereby constructing something that would resemble a map with coordinates.

7.2 Repairing The Repairers

Peter

If the Repairer agents all get disabled its hard to get them repaired and therefore also hard to repair the other agents. More focus should have been on getting the Repairer agents repaired immediately when they made a distress call.

7.3 Coordinated Repairing

Peter

When in distress the distressed agents could have met their assigned repairer half way, if not tied up in zone control. If the number of distressed agents are larger than the number of repairer agents, using the knowledge of where repairers would be at their next repair, the distressed agents could head toward these vertices. This would make it possible for the Repairer agents to quickly repair the other agents in distress.

7.4 Tracking Down The Opponents

Peter

In stead of randomly walking around, the Saboteur could have used knowledge on opponents found by other agents.

7.5 Prioritized Repairing And Attacking

Peter

Repairers could have prioritized which types of agents to repair depending on the total state of the individual group of agents and depending on whether in Zone Control mode or the mapping phase. Similarly saboteurs could prioritize repairers, saboteurs or inspectors. If successfully attacking a special group of opponent agents, the opponents would have a serious setback.

7.6 Making Proper Use Of Inspections

Peter

Primary concern of inspectors is to inspect the opponent agents as early in the game as possible. Having inspected an opponent agent once the only new information obtained inspecting an agent a second time is the health status. Our solution only inspects each agent once, primarily to get information on, which role the inspected agent is assigned. Subsequent inspections will only give new information if health status has changed after the agent has been attacked or repaired. The information regarding the agent role is not used further in our strategy.

Peter

7.7 Action Feedback

At the moment feedback of actions hasn't been used. Instead testing is done to see if there is sufficient energy to perform the planned action, and only actions allowed for the different agents are used. However knowledge can be obtained from the feedback of actions, eg. if an attack is being parried, we know the attacked agent is one of Repairer, Saboteur or Sentinel. Especially if feedback says the action has been prevented due to an attack, the agent is within range of a saboteur.

Peter

7.8 Opponent Strategy

There has been no attempt to detect the strategy of the opponent.

8 Conclusion

Andreas & Morten & Peter

In this report we have outlined how we have solved the problems stated in the problem analysis. We are able to gather knowledge about the environment during the Mapping phase. Here the agents' primary assignment is gathering more information about the environment where all the knowledge gathered is stored in a centralized knowledge base. The agents work together by virtue of the Planning Center, which will keep track of which actions the agents want to perform. The Planning Center will prefer the lowest cost action when two agents wish to perform an action that would give the same result. Further the Planning Center will not allow two actions that will achieve exactly the same thing. We use the knowledge base gathered from the Mapping phase when we enter into the second phase, Zone Control mode. We achieve a good score by using an algorithm that finds a connected component of maximum sum and tries to expand that component such that we dominate more vertices and create a high scoring zone. This was our strategy for the game. When we enter the second phase we are only being defensive. We do not try to counter our opponents in any way only in the first phase where we attack our opponents if possible. We are aware that there are flaws in our solution and these would be obvious choices if further improvement to our solution was to be done.

References

[1] Malik Ghallab, Dana Nau, Paolo Traverso: *Automated Planning: Theory & Practice*. 1st Edition, Morgan Kaufmann, 2004.

[2] Stuart J. Russel, Peter Norvig: *Artificial Intelligence: A Modern Approach*. 3rd Edition, Pearson, 2010.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: *Introduction To Algorithms*. 3rd Edition, MIT Press, 2009.

[4] Robert Sedgewick, Kevin Wayne: *Algorithms*. 4th Edition, Addison Wesley, 2011.