AASMA Project - Naive Cooperation

Marcos Pêgo - 86472
marcosppego@gmail.com

Marco Cabral - 87680
marco.cabral@tecnico.ulisboa.pt

ABSTRACT

In this report we will go in depth of the definition of our Project, Naive Cooperation, what approaches we took in its implementation and analyse our discoveries and findings.

Categories and Subject Descriptors

I.2.11 **[Computing Methodologies]**: Artificial Intelligence - Distributed Artificial Intelligence - Intelligent agents and Multi-agent systems.

General Terms

Algorithm, Design, Experimentation.

Keywords

Agents, A.I., Deliberative, Naive, Protector, Game

1. PROJECT DESCRIPTION

The project "Naive Cooperation" aims to simulate the relationship between two agents of different knowledge sets, using a multi-agent system design around a platformer game. The way we capture that relation is the following:

- There are two agents in the system that will try to complete the game as fast as possible. However each agent controls a character with different characteristics and capabilities.
- The first agent, called Naive agent, controls a blue sphere that can only see the end goal and the safe environment around it. This means that the Naive agent has no way of seeing or knowing about dangers around it by itself.
- The second agent, called Protector agent, controls a red cube that can change its shape out of three possible shapes. It is also capable of seeing all dangers and hazards that are spread around the world; however, unlike the Naive agent, it cannot see the end goal, and even if it reaches the end goal it cannot win the level.

To combat these adversities between the agents, there is a communication system in place that will help the agents coordinate between them so they aren't completely lost during the game playthrough. However, their communication system is not very advanced and has some restrictions: The protector agent can only convey the information of how dangerous the current state of the Naive agent is by giving it a danger level, this danger level is up to the protector to evaluate; The Protector agent can also instruct the Naive agent when certain actions should be performed, such as jumping or going forward; The Naive agent can inform the general direction of where the end goal is located but never its actual position.

1.1 Project Motivation

The main factor that led us to create this project was both of us majoring in Games alongside Artificial Intelligence and it seemed appropriate to work on an AI system to be implemented inside a game. This prompted us to think on what kind of agents and style of game we wanted to implement.

Eventually we came up with the idea of the Naive Cooperation Project and its interesting cooperation system.

We expect that by learning how to create dependency between agents like the dependency the naive agent has of the protector agent we can improve our skills in creating multi-agent systems. Given that the Video Games area has a lot of AI and multi-agent related fields, there are a lot of possibilities where we could use this skill in the future. For example, simulating biological ecosystems where animals depend on cooperation to survive or creating a complex companion AI system.

In short, we aim to learn to create different types of relations between AIs to better improve our future work.

1.2 Project End Goal

Our expected end goal for this project is to see if we can create, using a simple system of communication, a set of AIs competent enough to solve tiny platformer-like levels of varying difficulties. Can this simple system of communication paired with capable AI be enough to overcome the major setback of not being able to see the enemy or not being able to see the end goal?

We expect to answer this question by the end of the project.

1.3 Game Structure

The game is a platformer with the following characteristics:

- The floor and walls: consisting of different sized shaped cubes which form the overall world structure. This is where the naive and the protector agents will navigate in order to reach the end goal;
- The enemies/hazards: there are three kinds of enemies, a saw that spins in its place; spikes that pop in and out of the ground and a falling block that slowly goes up and then falls fast again, repeating the cycle;
- The end goal, a shiny green orb that represents the end of the level;
- The pit: the walls do not form a complete loop, which means that there is room for the players to leave the map, should they leave the map by accident they will fall towards the pit where they simply die.

1.4 Game Rules

There is only one way to win a level: the naive player reaches the end goal, and only the naive player, which means the protector player can't win the game without the naive player.

Both the naive and protector agents are restricted to the game's implemented movement system which means they can jump once after jumping off the floor and they can use left and right to move horizontally. They are both affected by gravity and they both collide with each other and the environment. When the naive agent moves it will spin on itself (like a wheel) towards the location, however the protector agent will only slide (not rotating) towards the location.

The naive agent is completely blind to the hazards around the world, the only allowed information comes from the protector agent in the form of their communication system. Similarly the protector agent is completely blind to the end goal, only being allowed information coming from the naive agent in the form of the communication system before mentioned.

The protector agent cannot be harmed by any hazard or enemy, will however die should it fall off the map by mistake. Should the protector agent die the game continues until either the naive agent dies or the level is won.

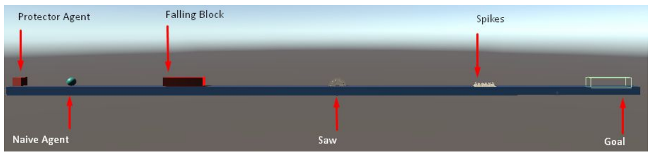
The naive agent will always die by any contact with enemies or hazards, like the protector it will also die should it fall off the map. If the Naive player dies, the game is over and the level is restarted.

Falling Block: at first contact between the protector agent and the falling block, the falling block will stop in its entirety until the end of the game. It will stop in the exact position where it was when the contact happened.

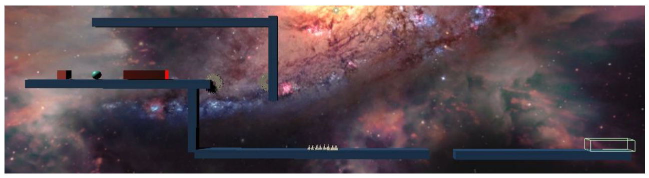
The saw and spikes will not change their behaviour throughout the game.

1.5 Level Format

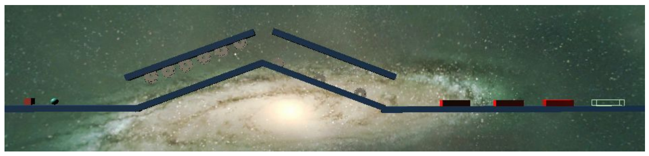
The first level is very simple and straight, featuring just one of each hazard.



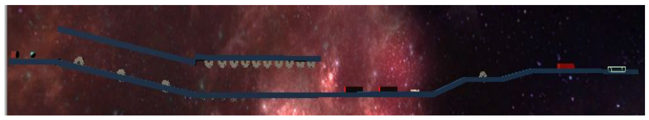
The second level is slightly harder. It makes the agents deal with saws in a vertical state and has a hole that the agents do not know how to deal with:



The third level features a slope, several saws in a row (that the agents must either ignore or jump over) and three falling blocks in a row:



The fourth level is a very long level and features even more slopes. It has a complete section to show that the agents are capable of ignoring dangers that do not pose a threat to the naive agent and a higher complexity in the second half of the level:



1.6 Naive Agent distrust

We decided to also analyse different ways the Naive agent could cooperate with the information given by the Protector agent. It could also be eventually complemented with the same for the Protector agent.

It consists in simulating mistrust by a given agent of the other. We aim to simulate this by using a given value **alpha**, which determines how often the Naive Agent trusts the Protector's judgment. The alpha represents a probability of deviation of standard cooperation with 0% being total cooperation and 100% being total disregard of the Protector's information.

With the possibility to define alpha values between 0 and 100 to see a possible threshold of how much distrust can the system handle and still win comfortably.

1.7 Result Expectations

Like mentioned before, 4 levels were developed with different and scaling difficulty. We aim to use the same AI we built in all 4 of them.

This will allow us to gage the flexibility and viability of the algorithms developed.

Given the scaling difficulty we expect around 100% win rate for the first two levels. They are fairly simple with little twists, 100% win rate here would mean the AI is at least capable of dealing with the hazards we implemented in the game.

For the last two levels we expect a win rate from 60 to around 90%. They are fairly more complex and require a better understanding of not only the hazards but the way they are displayed as well.

We consider a success if the agents for an alpha of 0 can soundly win all 4 levels at least once in a row.

For different valued alphas we had diverging opinions. One of us still expects the AI to be able to win with smaller valued alphas but never win with higher alphas. The other expects that regardless of the value of alpha, if it's any bigger than 0 the agents will always fail their task.

In the next topics we expect to have a proper analysis on what actually happened and why.

2. AI implementation

In this topic we will follow through possible implementations we could've used to implement our desired AI behaviour and why we ended up using our approach.

2.1 Why not Machine Learning?

Even though machine learning and reinforcement is a fairly interesting approach and can lead to some powerful results, it can lead to strange or unorthodox solutions that even though are probably more efficient they are not the most logical in the usual human thought process.

A good example is in the paper addressed in this video: https://www.youtube.com/watch?v=GdTBqBnqhaQ

The first topic addresses a virtual spider, and they gave an AI the goal to walk with it whilst keeping the foot on the ground for the least time possible. They gave it the goal of less than 10% of the time with foot contact in the ground.

The AI ended up finding it was possible to achieve with 0% contact time. By flipping itself upside down it could walk using its "elbows" and never use its feet to walk. Yes, this was for sure an amazing find and quite outside of the box, but we find it unhelpful if the goal is to create a realistic spider for a game or other possible applications.

We are aware though it would be possible to create a realistic looking spider with machine learning, but that would be equivalent of using a powerful tool and dumbing it down enough so we could have the desired effect.

Most game's AIs are created by behaviour trees and state management where they address different situations using its already predetermined and prebuilt protocol. Even though it does not create the most intelligent or most task efficient AI, it conveys a more logical approach where the human players can identify normal behaviours and actions.

2.2 Deliberative Agents

Looking at the subjects addressed in Aasma. The closest representation of a behaviour tree is the deliberative agent architecture, since we could implement the AIs with the logic we wanted.

We decided to use this approach in order to implement the agents' goals and actions. The desire was simple, the naive agent desires to win and the protector agent desires to ensure the naive agent wins.

We used mostly declarative programming which consists in: we say what we want to achieve, give the system info about the relationships between objects, and let a built-in control mechanism figure out what to do. This way the agents can adapt to several different maps we give them and still succeed in completing a level.

Further in the report we will go in detail on how we developed each built-in control system and how it helps the agents achieve what we want them to achieve.

2.3 Agent Movement - Map complexity

Movement around a 2D world can be fairly complex. Given we have physics implemented, walls and floor, it can be said that navigating this platformer can be much like navigating a labyrinth.

How do we know that going in the direction of the end goal will not result in a dead end?

Two possible solutions:

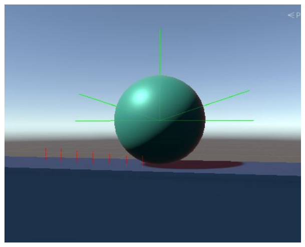
- Path-finding: with path-finding it's possible to predetermine a path which will guarantee the agents will not randomly stumble into a dead end;
- Simple Map Layout: In short, like a pipeline of water, the platform map always flows in the same direction, this allows the agent to always be somewhat sure that following the victory direction will not lead into a dead end.

We went with the latter approach for the simple reason that, by giving path-finding to the agents, we would be removing some of the dynamic approach to the possible obstacles the agents would find throughout the game.

Best possible solution: Hybrid movement, with the overall movement being calculated by a path-finding algorithm and in the micro level being addressed by an obstacle avoiding algorithm. However this would be fairly complicated and out of scope for two students to implement.

2.4 Agent Movement - The "Whiskers" method

Now we'll explain how we implemented a simple movement system, but at the same time complex, that allows the agents to cleverly address random obstacles that could appear during the movement towards the end goal.



We implemented a whiskers system, with 8 rays being cast outwards from the agent's body with limited length (the picture does not represent the actual length of the whiskers).

When the ray intercepts an obstacle, for example, the bottom whisker intercepts the floor in the picture shown above, it is disabled with the information that it's not possible to move in that direction.

This allows us to map a general direction of movement, we then use a movement function that takes a given direction and the rays still available (with no interception) and uses the ray with the smallest angle to the given direction.

After choosing which ray direction it will use, the agent uses the available movement system to try and achieve that direction.

This system allows the agents to dynamically address any visible obstacle that appears in "sight", this way we can change the map layout without affecting the agent's movement and without needing path-finding.

2.5 Naive Agent

2.5.1 Sensors

The Victory Direction: the naive agent is able to capture the direction between him and the victory point.

Raycast towards Victory: the naive agent is able to cast a ray towards the victory point, returns true if the ray hits the victory point and returns false if it intercepts anything in the way. This allows the naive agent to be aware if the victory point is within sight or if there is something in between.

Raycast in cardinal directions: Creates 8 raycasts around the naive agent with limited range. Like mentioned before these are the movement whiskers.

2.5.2 Actions

The agent has access to the following actions:

- Idle: this is useful if the agent has the need to stop completely (will still be affected by gravity). The agent will try to use the movement system to stop moving.
- MoveWithRayCast: the agent will use its whiskers to move in the given direction. By definition the direction the naive agent wants to move is the direction of the goal, therefore the agent cannot use any other direction.
- Jump: Although already incorporated in the MoveWithRayCast, there could be situations where simply jumping could prove useful.

2.5.3 Deliberation

The deliberation system for the naive agent is simple, it will move or jump if it has no other directive. With its cleaver movement it's enough to move around without walking into much constraint.

However like mentioned before, it cannot see the hazards so it will take into account information from the protector. Should there be any danger level, the naive agent will stop until the danger is addressed the best way possible (determined by the protector agent).

The alpha value determines how often the naive agent stops when there is danger level, or how often the naive agent keeps going.

2.5.4 Naive Communication

The Naive agent can send the direction of the victory point to the protector agent and nothing more. It can not send if it sees the victory point or the position of it.

2.6 Protector Agent

2.6.1 Sensors

PercieveNaiveDangerLevel: The protector agent has access to the naive agent's location; to determine the safety of the naive agent it casts rays from the agent to all known forms of hazards. If a direct ray is possible between the naive and the hazard then it means it is in its sight (not behind a wall for example). After determining which hazards are in sight it takes only in consideration the closest to the naive agent.

RayCastDanger: Sensor used to determine if the hazard is in the naive agent's line of sight.

Raycast in cardinal directions: Much like the naive agent, the protector agent uses the whiskers method to determine its movement.

AvoidNaive: if the agents are about to collide the protector agent gets signalled it should probably jump over the naive agent.

CheckOuterBounds: the agent will check if it has enough space to achieve its desired form, this avoids breaking physics like changing into a bigger block in a small space.

2.6.2 Actions

The agent has access to the following actions:

- Idle: this is useful if the agent has the need to stop completely (will still be affected by gravity). The agent will try to use the movement system to stop moving.
- OrientedMove: This function is just used to move the protector agent towards the given vector. It can't handle jumping. It's used by the protector agent to better position itself on top of the hazards.
- OrientedMoveWithRayCast: the agent will use its whiskers to move in the given direction.
- AvoidNaive: Function called when the naive agent is in front of the protector agent. It just makes the protector agent jump over the naive agent.
- ChangeShape: given a number, the protector tries to change its shape. It can change to its standard shape of a 1x1x1 cube, or to a 1x3x1 block or to a 1x1x3 block. In the later two shapes ir first verifies if the area surrounding it it's clear and therefore possible to change shape.
- Jump: the protector agent jumps.

It also has the following Compound Actions:

AddressSaw:

If the closest hazard to the naive agent is a saw, the protector agent will take the following steps depending on the saw's position:

- If the saw is above the naive agent then it gets ignored and the naive is informed that it should not jump; This is due to the fact that a saw above the naive agent poses no threat if the naive agent avoids jumping near it.
- If the saw is below or at the level of the naive agent the protector agent will try to move on top of it or between

it and the naive agent; This signals the naive agent to jump over the protector agent and by consequence jumping over the saw blade. This is a useful method of avoiding danger without the naive agent actually being aware where it is.

- Given the flow of the map, the protector agent also checks if the naive agent already passed the saw, if it did then it stops considering the saw a threat; This helps avoiding over analysing situations that are already resolved.

AddressSpike:

If the closest hazard to the naive agent is a spike, the protector agent will take the following steps:

- It will try to change its shape to better protect the naive agent, this is done by changing into a horizontal block that allows for more coverage of the spikes.
- Then it will try to be over the spikes covering them
- Like the sawblade it will signal the naive agent to jump either over or on top of the protector agent
- After the naive agent moves over the spikes into the other side, the protector agent stops regarding the spikes as a danger since they are already resolved.

AddressFallBlock:

If the closest hazard to the naive agent is a falling block, the protector agent will take the following steps:

- It will only try to move towards the block if it is already high in the air; This helps avoiding stopping the falling block too early and too low in the air.
- When the falling block moves up at a high enough altitude, the protector agent will try to make contact with it to stop it in its place.
- After the collision the protector agent will signal the path is safe and will try to move behind the naive agent to follow it along; This is mainly due to the fact that, if the protector agent falls in front of the naive agent, the naive agent will react by trying to jump over it. Usually the falling block is not high enough for the naive agent to jump and causes it to die.

2.6.3 Deliberate

After perceiving the expected danger of the naive agent, the protector agent will act accordingly:

If there is no danger the protector agent will try to reduce the distance between him and the naive agent whilst avoiding being in the way. This was done by ensuring the agent will never follow too close to the other.

If there is danger, then the protector will identify what is the nearest danger and act accordingly to each one. The protector agent will use each action available to best address each situation and when it feels there is nothing else it can do it will inform the naive agent the danger has been addressed.

This is because the agent can't be everywhere and can't cover all nearby dangers at the same time, so a compromise must be made and the protector agent must accept that there isn't anything else it can do even though it knows there are still dangers nearby

2.6.4 Communication

The protector agent can instruct simple tasks to the naive agent, such as asking it to jump. This allows for a more complex answer for the naive agent rather than just going forward like all danger is gone.

The protector player also has the ability to inform the danger level, in hopes the naive agent will react with weariness when the danger level is above 0 and it will stop completely. Giving the protector agent time to address the situation.

3. Road Blocks

In this section we will address the main issues we had during implementation of our AIs and the game world.

3.1 Problems found and our approaches

One of the main problems we had was the way the naive agent moved towards the goal. Since we didn't want to give the agent the capability of knowing the best path to the goal by giving him access to a path-finding algorithm, we were limited in the way we could create the levels. This means that we could not create levels that would either leave the naive agent having to choose between going left or right when both options were ideal in its point of view or would make the agent go in the opposite direction of the goal .

One of the situations we found constraining was when we tried to build a vertical kind of platformer level. Since the victory point was directly downwards the AI wouldn't know whether going left or right would bring it closer to the goal, causing it to stutter between each direction since moving any direction would cause it to stray further than where he already was.

Like already mentioned before, we ended up choosing a more linear direction of map, where all the map flowed into the same direction.

Another problem we had was with the shape of the protector agent. We decided for it to be a cube in order to better protect the naive agent, however this resulted in a more restricted movement, meaning the agent would often get stuck in some hazards or slopes and would take a while for it to get unstuck. We tried giving the agent the liberty to rotate on the x axis, but this just made the agent have a very chaotic movement which would result in the loss of the level.

We decided to maintain the shape and try to minimize the situations where the agent would get stuck not because of its incapability but because of our world/physics implementation.

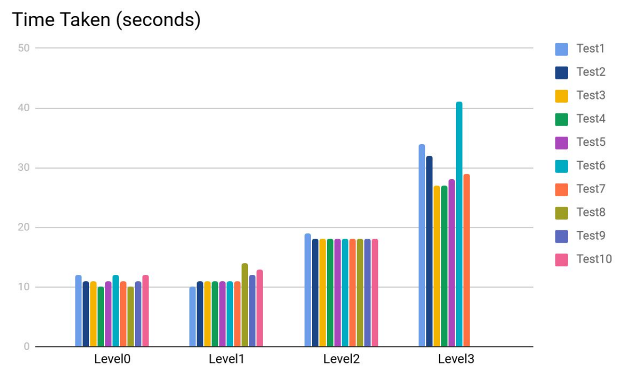
4. Experimental Analyses, discussion and findings

In this section we will talk about the testing and results we obtained during testing. At the end of the section we'll discuss the overall project result.

4.1 Experimental Analyses

In order to evaluate the performance of our agents we decided to guide by two metrics: times succeeded/lost in 10 runs and the time taken. We then applied these metrics to 3 different alpha values (0, 50, 100) and got the following results:

Alpha = 0			
	Success	**Stuck**	**Lost**
Level0	10	0	0
Level1	10	0	0
Level2	10	0	0
Level3	7	0	3



Alpha = 0	
	Average Time (seconds):
Level0	11.1
Level1	11.5
Level2	18.1
Level3	31.14

Alpha = 50			
	Success	**Stuck**	**Lost**
Level0	0	0	10
Level1	0	0	10
Level2	0	0	10
Level3	0	0	10

Alpha = 100			
	Success	**Stuck**	**Lost**
Level0	0	0	10
Level1	0	0	10
Level2	0	0	10
Level3	0	0	10

4.2 Findings

Although there's a small number of tests for each level, we believe it's enough to reach the conclusions we wanted.

Firstly regarding the alpha: with the alpha at 50 or 100 the agent was never able to complete a level. Ignoring the protector agent would always lead the naive agent towards its doom. We tried testing out of curiosity with the very low alpha value of 5, however even then the naive agent would get himself killed more often than not.

Then regarding the efficiency of the agents. We can see that in the first three levels the agents were able to succeed the 10 runs and maintain a time very close to the average. In the last level the agent was able to succeed in 7 out of 10 runs, but did not maintain a time very close to the average. We can conclude with this that the agents are very effective and able to complete levels in a very reliable way, while at the same time we can notice that, by looking at the time differences of level3, the agents still have to adapt to a level the way it's given to them and that each run will end up being different from the one before.

4.3 Discussion

4.3.1 Why didn't the untrusting agent have any success?

The obvious answer is given the way the project was built, the naive agent is at a high disadvantage without the help of the protector agent. He can not see any kind of dangers and is killed in one hit, so any mistake results in a fail.

There's also the way that the hazards were made. Although it would be possible to pass through the falling blocks and the spikes with perfect timing, a saw blade on the ground would always result in the naive agent's death.

4.3.2 So how can an agent that ignores its only reliable source of information have any chance of success?

The simple answer is it can not. However the key word is **reliable** source.

What if the source was not reliable? What if the protector agent also had a conflicting interest where maybe he wanted the naive agent to win but in the maximum time possible? Well then the naive agent could in theory assess the reliability of the information and sometimes ignore it in its entirety. In short, not trusting the protector agent because it in fact did not have the naive agent's best interests completely ensured.

The reality is that the protector agent we implemented will always try, without fail, to ensure the safety of the naive agent, so there is no realistic reason to disregard its warnings. Maybe the way the protector agent approaches each situation is not the most efficient but the naive agent has no choice but to accept what it has, for no other reason than it can't win on its own.

4.3.3 Why didn't we get a 100% win rate in all maps?

The last map addresses a flaw in our system, which it's hard to fight. Timing.

The seemingly useless corridor of nothing highlights the arbitrary factor of our unity built system. The world is not constant and each iteration has a different outcome. The big corridor makes for a bigger difference in positioning when the hazards do appear, highlighting some flaws of the "mechanized" reactions of our agents that lead to agent mispositioning and eventually death.

4.3.4 Why didn't we address this flaw?

For the simple fact that in our understanding would always be a flaw of a non static world, each action could have different outcomes and even if we follow the same set of actions constantly the result could always be different. We could spend our entire time trying to make the system more "robust" but since we are using deliberative agents unless we have a perfect set of actions for each state we could always devise a map where the AI would struggle to constantly have the right approach.

4.4 Conclusion

Our final conclusion is the following:

We envisioned a group of AIs that had to cooperate in order to win a platformer game and we think we achieved just that.

The AIs are not too complex, but are certainly not simple. They can address different situations with varying levels of difficulty by just applying the concepts lectured in this class like "means-to-end reasoning".

During our development we also questioned ourselves if it was possible for the naive agent to sometimes disregard the protector agen's advice and reached a fairly solid conclusion.

We would also like to try a shady protector that would sometimes mess with the information but unfortunately this was way out of what 2 students could do on top of the already existing project.

Overall we are fairly happy with the results, and watching the tiny box and ball strolling around avoiding danger by themselves is satisfying and gives a real sense of Artificial Intelligence.