INGI1131 – Project 2011

Gustavo Gutierrez Sébastien Doeraene

December 2, 2011

1 Introduction

It is fall and outside activities are a bit more difficult everyday because of the lack of sunlight. Interestingly, this is not a problem for the students of *INGI1131* because fall means time for the project! This time, we decided to come up with something new, but challenging as always in the history of this course.

This document describes the project of the course. You have to read it very carefully and then to propose your solution. For doing the project there are some rules and important aspects that you should consider.

1.1 Deadlines

You have two weeks to make the project. Douring this time the assistants will be willing to help you and guide you in any problem you have. Consider the following strict deadlines:

November 26 at 11:58 pm Beginning of the project.

December 9 at 11:58 pm Deadline for the submission of your project.

December ? Small contest between your brains (AIs) and some prizes. Continue reading the document for more details. Precise date yet to decide, but will be after the 9th.

December 25 Merry christmas!!

2 A world driven by economy and concurrency

At first sight, the title looks weird, but yes, it is true, or at least for this project. We live in a world where economy plays a very important role in society. Actually the following context is not that far from reality.

There are different societies in this planet, for the effects of the project we will refer to them as *teams*. Members of the societies are normal people like you and me and we will call them the *players*. In order to survive as a team, every member has some tasks that contribute to the final goal: survive.

To survive we need to be healthy and have food, among others. We get this by exploiting the resources that we have in our planet. For instance, in order to be warm (and not ill) during winter we need wood. That wood is considered a resource and can be exploited in different places in our planet: the forests. This is only an example of a *resource* that we *need* and that we *exploit*.

Of course, our team is not the only one that needs wood so we need to exploit and sometimes fight for exploiting it. Remember, wood is not the only resource, it is just an example. The idea here is to make clear that societies *compete* for resources.

2.1 Social behavior

There are some important aspects of the social behavior that you should consider. Even if at some point they do sound very philosophical they will be represented in your project.

2.1.1 Societies are not nomads

This is straightforward to understand, we live in houses, inside cities. We need the resources to be there, at our disposal. We will say that every team has a basecamp, i.e, a home. Every individual needs to exploit resources and bring them back to the basecamp.

2.1.2 Resource exploitation

Resources exist in several parts of the planet but not everywhere. That means that it is very likely that several teams become interested in exploiting the same resource. It is only then when things can become messy.

A hopefully big difference with the real world is that people from different teams are quite selfish. That means that they do not want to share the resources with anyone else. When different players try to exploit the same resource in the very same place the resource exploitation will depend on the *strength* of all the interested players. Only the strong players will succeed and the others will die. In a further section we will explain this concept of *strength*.

Players can build *towers* to watch and secure parts of the world. A tower will defend a specific area by killing every individual that dares to approach. Of course, a player of the same team that placed the tower has free transit in the secure zone established by the tower. Building a tower has a price, so teams will have to pay for it. As this economy is driven only by resources, paying for a tower will impact the resources that a team has accumulated in its basecamp.

A player cannot exploit a resource forever. Remember that the goal is to bring resources to the basecamp. When a player is exploiting a resource, it has to take into account how many units he can transport to the basecamp. We call this the capacity of the player. When the capacity of a player is full it cannot exploit the resource anymore. Only after going to the basecamp and depositing all the resource units he has collected it is possible to come back and continue the exploitation.

2.1.3 Interaction with other societies

In the previous section we already explained how several societies interact when exploiting the same resources. There is another way of interaction though. As basecamps have a place in the world they can be seen as *resources* for other teams. As this is a selfish model we are not talking about sharing resources but stealing them.

Players of a team can go to other basecamps to steal resources. As basecamps behave like resources, it is possible to fight or protect them (for instance by putting towers).

2.1.4 About life and death

One of the resources that can be exploited is food. This is probably one of the most important ones because it contributes to life in a very particular way. As players can be killed new players can be created too. This has a cost and affects the food resources of the team. When a player is created, it appears at the basecamp of the team and the food savings of the team decrease.

In order to protect its life and the goods collected by its team, players can acquire a weapon. This will serve when the player meets other players of different teams. By having a weapon the player increases its strength but it comes at the cost of spending some resources.

2.1.5 Long term preservation

2.2 Individual behavior

When players are in a direct confrontation for a resource, they fight with others. The winner or winners are the strongest ones. Let's start by the simplest case of a confrontation when two players compete for a resource. If both of them have the same strength then both get dead. If one of them has a weapon then it is considered stronger and the other player dies.

When there are more than two players in a confrontation, the winner are the players of the team with the greatest strength. This strength is computed by the addition of all the individual strengths of the players of the team. Under this rule, all the player of the weakest team die.

3 Precise rules

Since there is going to be a tournament, you will need precise rules, i.e., specifications of the game. This section aims at providing you with such details.

If at some point there is contradiction between this section and the previous one, this section prevails.

Note: all the numerical constants that are mentioned in this section should not be considered as sealed. The figures we gave are just "reasonable" defaults that the authors thought good. It is very likely that you will find that some figures are not suited for a balanced game. We encourage you to share on iCampus any amendment that you judge appropriate.

3.1 A real-time game

The game is *real-time*, very much like Age of Empires. Several players move independently of the others. If one player decides his actions faster than another, he will be able to take an advance with respect to the other players. So it is *not* a turn-based game.

3.2 Teams, players and resources

There are several *teams* (at least 2), and each team has zero to many *players*. Players walk on the board. They can die and new players can be created. Each team has a *home* which is a square on the board.

There are 4 kinds of *resources*: food, wood, stone and steel. Each team has a given amount of each resource (non-negative integer) that it keeps safe at its home. Resources at a team's home can be used to pay builds, or can be kept in order to reach the given goal.

3.3 Goal

The goal of a team is to be the first team to gather at least a given threshold of all resources at its home. The thresholds should be configurable, and can be different for each resource. A recommanded default is 200 units of all resources.

3.4 Board

The game is playing on a *board* of discrete squares. There are `Width` × `Height` squares on the board. Each square of the board has a *kind* among the following.

Normal square: A normal square, without resource or home on it.

Field: A square that provides food.

Forest: A square that provides wood.

Quarry: A square that provides stone.

Mine: A square that provides steel.

Home: The home of a team.

Note that the kind of a square does not change during the game, i.e., resources and homes do not move or disappear. Squares that provide a resource are called *resource providers*.

There *can* be several players on the same square, even from different teams. Players do not attack each other when they are on the same square (they do so only when they try to exploit the same resource, which is discussed later).

3.5 Moving and collecting resources

At any time, a player can decide to move to one of the 8 squares surrounding him. This movement takes one time unit (usually 1 second) to happen.

When a player is on a resource provider, he can decide to *exploit* the resource instead of moving. This takes 1 time unit, and the result is that it receives 1 unit of the resource he is exploiting. This resource is not added to the team's resource pool automatically. Instead, the player *holds* that resource in his bag. A player's bag can contain several units of different resources, but it has a maximum *capacity* of 10 units in total. In other words, a player can hold 4 units of wood and 6 units of food, for example. If a player whose bag is full tries to exploit a resource, a time unit elapses but it is lost, as the bag cannot contain more resources.

Several players of the same team can exploit the same resource provider at the same time, each of them receiving 1 unit of the given resource. For players of different teams, this triggers a *fight*, as discussed in Section 3.10.

When a player is on his team's home, he automatically deposits all the resources he is holding in his bag into his team's home. The bag is then emptied, and the resources are available either to reach the goal thresholds or to build something.

When a player is on an opponent's home, he can *steal* from that home the resources that are stored there. When doing this, an opponent's home acts as any other resource provider, except that it has limited supplies, and that the player must tell the system which resource he wants to steal. In order to defend the home from opponents, two towers are initially built next to each home (see Section 3.11).

3.6 Builds

Instead of moving or exploiting a resource, a player of a given team can use the resources available in his team's home to *build* things. Again, one time unit elapses between the moment the build is commanded and the actual build. Each thing has a cost in resources. This amount of resources must be available *in the team's home* (not in the player's bag) for the build to succeed, at which point those resources are discarded. Some builds have *prerequisites*. If these are not fulfilled when the action is to be performed, the build is *cancelled*, and the resources if costs are *not* discarded.

3.6.1 Player

Cost: 10 units of food.

Parameters: the initial state for the player's brain (see Section 3.7).

Prerequisite: none.

Pop a new player at the team's home. The initial state of this player's brain is specified by the build command, i.e., it is chosen by the acting player. It is the state of the *brain*, so it is totally up to you (as a designer of the brain) to specify it, i.e., it is not related to the game concepts, but how you plan to make your brain behave. See Section 3.7 for details about brains.

3.6.2 Weapon

Cost: 25 units of steel.

Parameters: none.

Prerequisite: the acting player does not hold a weapon yet.

Provide the acting player with a *weapon*. A weapon strengthens a player when he has to fight (see Section 3.10).

3.6.3 Tower

Cost: 50 units of wood and 50 units of stone.

Parameters: coordinates (x, y) of the target square where the tower is to be built.

Prerequisite: the target square is next to the acting player (diagonals excluded), it is a *normal* square, and there is no tower yet on that square.

Build a tower on the target square (x, y). Towers are used to defend an area from foreign players. More details about towers in Section 3.11.

3.7 Brains

Each player on the game is associated to a *brain*, which is a software component that acts as an AI for that player. The brain can maintain an implicit state (an accumulator), but this state is local to one player. In other words, different players, even from the same team, cannot communicate.

The brain is what will be evaluated by the *contest*, and will not be graded. The rest of your program is what will be *graded*.

3.8 Environment (input of the brain)

In order to take useful decisions, a brain has to get some knowledge about the current state of the game. Upon creation of a new player, a corresponding brain is created as well, and is given all the static details about the board. Hence, a brain knows the kind of all the squares on the board; how many teams there are and what team it is part of; the goal thresholds of resources.

Each time the brain is called by a player in order to make a decision, it additionally receives an *environment* which contains some (but not all) of the dynamics of the game. The environment contains the following information:

- The current location on the board of the controlled player
- The contents of the controlled player's bag
- Whether the player has a weapon or not
- The amount of resources the player's team has currently in its home
- If the current square is an opponent's home, the amount of resources stored at this home
- The location and owner of all towers that are *visible* from the player's current position (see Section 3.11)

3.9 Actions (output of the brain)

When given a state and an environment, the brain must reply with a new state and an *action*. There are four kinds of actions.

No-op , i.e., do nothing. This action does not take a time unit to execute.

Move to one of the 8 surrounding square.

Exploit the resource provider on the player's square; obviously valid only if the player is on a resource provider or an opponent's home.

Build something.

3.10 Fights

When a player tries to *exploit* a resource provider that is already being exploited by at least one player of another team, this exploitation is interrupted (cancelled), and all the players that were previously exploiting the resource pick up a *fight*.

So, given a time unit of $1.000ms$, let's say at time t a player from team A starts exploiting a resource. If, at time $t + 300ms$, a player from team B starts exploiting the same resource, a fight starts, since the first player is still exploiting the resource.

At that point, the *strenghts* of both parties are computed. The strenght of a player without a weapon is 1, and the strenght of player with a weapon

is 3. The strenght of a team is the sum of the strenghts of all players from that team that are part of the fight. If the computed strenghts are equal, then all players involved in the fight die (i.e., disappears from the map). If one strenght is greater than the other, then all players from the losing team die and the players from the winning team stay alive, and become idle (i.e., they must take a new action).

3.11 Towers

The purpose of a tower is to defend an area, so that foe players cannot enter it without dying. A tower has a *range*, which is 2. That means that it covers any square which is at a distance smaller or equal to 2. This distance is defined by $\max(|x_t - x_p|, |y_t - y_t|)$. The set of squares in this range are called the *area* of the tower. When a player enters a foe tower's area, he dies immediately.

When a tower is built, it has initially 20 resistance points. Each time a player enters its area, his death is not in vain, as the resistance points of the tower decrease by an amount equal to the player's strenght. When a tower has no more resistance points, it is destroyed.

If a player enters the area of more than one foe tower, then only one of those towers looses resistance points (the choice of which one being left unspecified).

The resistance of a tower cannot be increased.

Towers are *visible* by players in a given area. That area covers the squares whose distance to the tower is smaller or equal to 4, the distance being defined as above.

3.12 Initial state of the game

We recommand that the *statics* of a game (positions of the homes and resource providers, goal thresholds, etc.) be stored in a file that describes the map. That should be much easier than to generate maps at random in your program, and also more useful, as you can obtain reproducible results.

Each team begins with no resource in its home, and with 5 players, all located in the home square. The home of a team is initially protected by two towers, that are at opposite diagonal edges of the square.

The initial state of the initial players' brains is `none`.

4 The actual project statement

OK, now that you have all the information about the game context and precise rules, here is the actual project statement.

You are asked to implement a *simulation* of this game. The program should not be interactive, i.e., players will act by themselves without requiring user input. They are controlled by their brain.

Note that your program is supposed to work with any number of teams ≥ 2. As previously stated, we recommand loading the game map from a file. You can have a look at the `Open` module of Mozart to read a file.

We encourage you to provide a graphical user interface (GUI) for your program. This GUI should be only an *observer* of the simulation. You will find on iCampus bootstrapping information to build user interfaces in Oz with QTk. However, please keep in mind that the project is not about GUI design, but about *concurrency*. Hence, the GUI is supposed to be really minimal. You can obviously use object-oriented code for the GUI module, as a GUI is intrisically mutable. You must not have explicit state outside of this module.

In order to be successful in this project, you will need many of the concurrency concepts that you learned in the course. It is forbidden to use any form of explicit state in the program (cells and classes) outside of the GUI module.

Advice. Think before you type! Think carefully about the architecture and design of your program. A good design will be most valuable and will make your life easier during the development.

Another advice. Implement the project incrementally. It is possible to identify subset of the rules that can be implemented first. For example, you can consider this really core rules: players have to exploit resources and bring them back to their home to reach the goal thresholds, i.e., no fights, no builds, no stealing in opponents' homes. Once you have this core implemented and functional, you can start expanding your implementation towards the full rules. A next-to-core set rules would include *fights*. **Please follow this advice! It is a *very* valuable one!**

Implementing correctly and nicely the rules up to fights (without builds and stealing in other opponents' homes) is already worth acceptable grades for the project.

5 Tournament

For the tournament, your brains and programs must be interoperable. In practice, this means that the interface between the brain and the program must be standardized. We will not do this standardization for you, so you must take the initiative and discuss about it on iCampus.

We recommand, though, that the brains be stand-alone *functors*, that can be loaded statically or dynamically by your program. See the Mozart documentation, and two extra set of exercices of the two first lab session for information about functors.

6 Deliverables, aka practical details

This project must be performed by groups of maximum *two* students. In order to submit it you have to send an e-mail to `sebastien.doeraene@uclouvain.be` and `gutierrez.gustavo@uclouvain.be`. This e-mail must have as subject `[INGI1131] lastname1-lastname2` and must contain as attachement an archive file in zip or tgz format whose name is `lastname1-lastname2.(zip|tgz)`. Of course, replace `lastname` with your last name. This archive must contain the following items:

- Full source code of your program.
- A basic brain that enables to run your program (there is *no need whatsoever* for your brain to be smart at all).
- A small report of about 2 pages, explaining the architecture and design of your project, along with the concurrency issues you faced and how you solved them. The full names of the authors must appear on the first page of the report. The report must be in .pdf format. We will *not* accept reports in .doc or .odt formats.