LINFO1131 - Project 2023
PacmOz: Escape The Matrix

Damien Sprockeels Raphaëlle Wats

November 16, 2023

1 Introduction

Your mission, should you choose to accept it, will be to implement concurrent agents operating within the 'PacmOz' message-passing protocol. These agents will engage in team-based battles within the virtual maze better known as 'The Matrix'.

You will be tasked as well to provide a Game Controller that ensures agents seeking access to 'The Matrix' adhere to the 'PacmOz' protocol and the rules of the Maze thus preventing it from falling into malevolent hands.

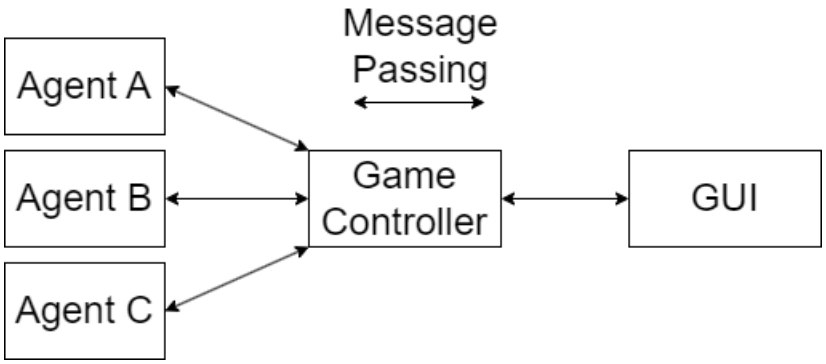
Please note: this document won't self-destruct in five seconds.

2 Rules of PacmOz

The game starts with the building of the Maze and the spawning of all the pacgums (points, labeled 0) and the pacpows (bonuses, labeled 2). Then, the PacmOz(s) and the GhOzt(s) spawns at one of the possible positions indicated in the Input.oz file.

- At the start of the game, pacmOz and ghOzt agents spawn in one of their possible spawning tile specified in the Input.oz file.
- All agents can share the same walkable square (labeled with a digit different than 1 cf. Input.oz).
- GhOzt agents will eliminate PacmOz agents if they are on the same square and a bonus isn't active.
- PacmOz agents pick up the pacgums/pacpow by walking on a tile where they are present.
- Each pacgums grants 100 points to the final score of the PacmOz agent team.
- When a pacpow is triggered, then all the ghOzt becomes vulnerable during a lapse of time of 3 seconds and PacmOz agents are able to eliminate them while they share the same square. Please note: Triggering another pacpow will reset the lasp of time.
- Pacpows respawn 10 seconds after being triggered.
- An eliminated GhOzt grants 500 points to the final score of the pacmOz that eliminated it.
- The PacmOz team wins the game when either all GhOzt agents are eliminated or all the pacgums are collected. If the PacmOz team wins the game then they achieve the maximum score (totalP acgum x 100 + totalGhOzt x 500).
- The GhOzt team wins if they achieve to eliminate all the PacMoz agent before they collect every pacgum of the Maze. Their goal is thus to minimize the score of the PacmOz team.
- An eleminated pacmOz or GhOzt agent doesn't respawn for the rest of the game.

3 Game general behaviour



The Game Controller will be in charge of communication between the Agents and the GUI. Every message must go through the game controller, even when an agent wants to send a message to another agent. It must make sure that every action attempted by the agents are valid, and inform other agents when necessary. It must also update the GUI with the relevant information when necessary.

4 Specifications

The following subsections will give you the specifications for all the agents you have to implement. We ask you to follow these in order to make your agents interoperable between each other. Please note that if you don't follow these rules, the automatic tests used to grade your project will not work.

4.1 Types (EBNF formula)

To ease the modularity, we define some types to respect.

```
<id>     ::= <int>
<type>   ::= pacmoz | ghozt
<bot>    ::= bot(<type> <atom> <int> <int>)
<direction> ::= north | south | east | west
```

4.2 Parameters of the game (`Input.oz`)

- bots: Is a list of $< bot >$ to spawn on the maze together with their type and spawn position.
- PacPowDuration and PacPowRespawnTime are respectively the duration and respawn time of the bonuses.
- genMaze: Is a function that returns a list of digit (0 = walkable square, 1 = wall, 2 = pacpow spawn). The generated Maze must be of dimension $28X29$. Please note: if your bot is at the position x = 1 and y = 1

```
Ncol = 4
Nrow = 3
Maze = [
    1 1 1 1
    2 0 4 1
    1 3 1 1
]

Then {Nth Maze Y * Ncol + X} + 1$ will output 0
Then {Nth Maze (Y - 1) * Ncol + X} + 1 will output 1
Then {Nth Maze (Y + 1) * Ncol + X} + 1 will output 3
```

```
Then {Nth Maze Y * Ncol + X + 1} + 1 will output 4
Then {Nth Maze Y * Ncol + X - 1} + 1 will output 2
```

4.3 Graphical User Interface (`Graphics.oz`)

The `Graphics.oz` file exports the function SpawnGraphics which launches the window and returns an instance of an active object. This file is provided to you. SpawnGraphics takes two arguments, the port of the game controller where it has to send messages back and a Tickrate. The active object can handle the following messages:

- `buildMaze(Maze)`: Iterate over the list of digits and buffer the background sprite to be displayed. For each 0 it will send back on the port the message $spawnPacgum(< int >< int >)$ and for each 2 it will send back on the port the message $spawnPacpow(< int >< int >)$.
- `spawnBot(Type X Y Id)`: Spawn a bot and returns a unique identifier, it will send back the message $movedTo(< id >< type >< int >< int >)$.
- `dispawnBot(Id)`: Remove a bot from the GUI.
- `moveBot(Id Dir)`: Move a bot to an adjacent square of the Maze. When the bot is done moving it will send back the message $movedTo(< id >< type >< int >< int >)$.
- `spawnPacpow(X Y)`: Spawn the pacpow on the GUI and send back the message $pacpowSpawned(< int >< int >)$.
- `dispawnPacpow(X Y)`: Remove the pacpow temporarily from the GUI, wait 5 seconds then send back the message $pacpowDown()$. Wait again 7 seconds then send back the message $pacpowSpawned(< int >< int >)$. Updates the sprite of the GhOzts team.
- `setAllScared(Boolean)`: Sets all the sprite to the scared version. false -¿ red, true -¿ blue. Must be called whenever pacpowDown() is received and the number of active pacpow is 0.
- `updateScore(Score)`: Takes an ¡int¿ as argument and update the score displayed by the GUI.

  Remark: The grid displaying the board works as a mathematical matrix. The top left square is $(0, 0)$, going from left to right increases the column number, going from top to bottom increases the row number. Square (X, Y) is at the intersection of the X^{th} column and Y^{th} row of the grid.

4.4 PacmOz agents and GhOzt agents (`PacmOzXXXname.oz and GhOztXXXname.oz`)

PacmOz and GhOzt are concurrent functional agents that are spawned in the Maze by the Game Controller, that is in charge of making sure no illegal move is performed by the agents. They must be spawned by calling the exported function under the label 'getPort' which takes a record $init(< id >< port >< maze >)$ as argument. To implement these agents, a basis is provided in the file `Agent000template.oz`. You must rename this file appropriately using the type of agent and the number of your group. The Id is given by the GUI, Port is the Game Controller. The agent must handle all the following messages, and should never crash due to unrecognized messages:

- `movedTo(Id Type X Y)`: The Game Controller broadcasts $movedTo(< id >< type >< int >< int >$ to every agent in the Maze. The agent can now send back to the Game Controller their next move with the message $moveTo(MyId < direction) >$. The agent must wait for the next $movedTo$ message where Id is equal to their id before sending back another moveTo message.
- `gotHaunted(Id)`: The Game Controller broadcasts $gotHaunted(< id >)$ to every agent in the Maze to signal that a PacmOz agent is eliminated from the game.
- `gotIncensed(Id)`: The Game Controller broadcasts $gotIncensed(< id >)$ to every agent in the Maze to signal that an GhOzt agent is eliminated from the game.

- `pacgumSpawned(X Y)`: The Game Controller broadcasts $pacgumSpawned(< int >< int >)$ to every agent in the Maze to signal that a pacgum has spawned at the following location.
- `pacgumDispawned(X Y)`: The Game Controller broadcasts $pacgumDispawned(< int >< int >)$ to every agent in the Maze to signal that a pacgum has dispawned at the following location.
- `pacpowSpawned(X Y)`: The Game Controller broadcasts $pacpowSpawned(< int >< int >)$ to every agent in the Maze to signal that a pacpow has spawned at the following location.
- `pacpowDispawned(X Y)`: The Game Controller broadcasts $pacpowDispawned(< int >< int >)$ to every agent in the Maze to signal that a pacpow has dispawned at the following location and a pacpow is now active.
- `pacpowDown(X Y)`: The Game Controller broadcasts $pacpowDown(< int >< int >)$ to every agent in the Maze to signal that one of the pacpow is no more active.
- `tellTeam(Id Record)`: The Game Controller broadcasts $tellTeam(< id >< record >)$ to every member of the agent's team whenever the agent sends back to the GameController the $tellTeam(MyId < record >)$ message. You are free to send a reasonable number of $tellTeam$ message to the Game Controller in order for the team to coordinate or not using your very own message-passing protocol. (Provide an explaination of the protocol used in the submitted report).
- `haunt(MyId PacmOzId) / incense(MyId GhOzId)`: GhOzt / PacmOz can send back those message to the Game Controller. If the two Id given describe agents sharing the same square then the second Agent is eliminated from the game if the eliminating conditions are met.
- `shutdown()`: Signals that the agent is eliminated thus can no longer send messages to the Game Controller. The agent must terminate and be cleaned by the garbage collector.
- `invalidAction()`: Signals the agent that the action it tried to perform is invalid (moving into a wall, trying to kill an agent that is not on the same tile,...).

4.5 Game Controller (`Main.oz`)

This file contains the main core of the project. This is the file that is responsible to launch the game and coordinate the agents to ensure they follow the rules. The Game Controller must be implemented as a concurrent functional agent.

This file should handle the following tasks:

1. Instantiate the GUI active object.
2. Instantiate the agents using the `AgentManager.oz` and assign its unique Id given by the GUI.
3. Launch and coordinate the game.
4. Check that agents behave respecting the rules.

The Game controller must handle the following messages, if the message is invalid it must simply discard it. It should never crash due to unhandeld messages:

- `moveTo(Id Dir)`: The Game Controller must check if the move is legal then send the $moveBot$ message to the GUI. The message $moveTo(< id >< dir >)$ is sent by either a GhOzt agent or a PacmOz agent. It must also broadcast the message to all agents.
- `movedTo(Id Type X Y)`: The Game Controller broadcasts $movedTo(< int >< type >< int >< int >)$ to every active agent in the Maze to signal that an agent is now in a new square of the Maze. The message is sent by either a GhOzt agent or a PacmOz agent. Before broadcasting to other agents, the game controller must make sure the move is valid. If the agent sending the

message is a pacmOz, the game controller must also check if it is on a tile where a pacgum/pacpow is present and broadcast the corresponding message. The score must also be increased if the pacman ate a pacgum.

- `haunt(PacmozId GhOztId)`: The Game Controller broadcasts $haunt(< int >< int >)$ to every active agent in the Maze to signal that a PacmOz agent has been killed. The message is sent by a GhOzt agent and the Game Controller must check if the eliminating conditions are met. The Game Controller then sends the $shutdown()$ message to the eliminated agent.

- `incense(PacmozId GhOztId)`: The Game Controller broadcasts $incense(< int >< int >)$ to every active agent in the Maze to signal that a GhOzt agent has been killed. The message is sent by a PacmOz agent and the Game Controller must check if the eliminating conditions are met. The Game Controller then sends the $shutdown()$ message to the eliminated agent.

- `pacgumSpawned(X Y)`: The Game Controller broadcasts $pacgumSpawned(< int >< int >)$ to every active agent in the Maze to signal that a pacgum has spawned at the following location. The message is sent by the GUI after it receives the $spawnPacgum$ message.

- `pacgumDispawned(X Y)`: The Game Controller broadcasts $pacgumDispawned(< int >< int >)$ to every active agent in the Maze to signal that a pacgum has dispawned at the following location. The message is sent by the GUI after it receives the $dispawnPacgum$ message.

- `pacpowSpawned(X Y)`: The Game Controller broadcasts $pacpowSpawned(< int >< int >)$ to every active agent in the Maze to signal that a pacpow has spawned at the following location and a pacpow is now active. That message is sent by the GUI 10 seconds after it receives the $dispawnPacpow$ message.

- `pacpowDispawned(X Y)`: The Game Controller broadcasts $pacpowDispawned(< int >< int >)$ to every active agent in the Maze to signal that a pacpow has dispawned at the following location and a pacpow is now active. That message is sent by the GUI after it receives the $dispawnPacpow$ message.

- `pacpowDown(X Y)`: The Game Controller broadcasts $pacpowDown(< int >< int >)$ to every agent in the Maze to signal that one of the pacpow is not active anymore. That message is sent by the GUI 3 seconds after the GUI received a $dispawnPacpow(< int >< int >)$ message.

- `tellTeam(Id Record)`: The Game Controller broadcasts the message $tellTeam(IdRecord)$ to every agent of the team's Id except the sender.

If an action performed by an agent is not allowed, the game controller will respond with an invalidAction() record.

5 Interoperability

If you follow the specifications, you will be able to test your implementation with other pacmOz teams and ghOzt teams from other groups. We ask you to test your pacmOz with another group's ghOzt, and your ghOzt with another group's pacmOz.

When sharing your team (ghOzt/pacmOz) with another group, you can't share the code! You first have to compile it on your side and give only the `.ozf` compiled file to the other group so they won't access your code (as sharing code is forbidden).

The list of the teams you have tested should be given in the report with remarks on how it went, if it helped you find mistakes,...

6 Submission and logistic details

6.1 Functors and compilation

For Syntax and use of functors, consult the book pages 220–230.

- Compiling functors : `ozc -c myfunctor.oz`
- Executing functors : `ozengine myfunctor.ozf`

A Makefile is provided to you, the `all` target will compile all functors and the `run` target will run the Main functor. It is only guaranteed to work on Linux, but should also work on MacOS. We ask you to submit your code with a Makefile that compiles on Linux as well. Your code should be able to run on the computers of the INTEL room, but you are free to work on any OS you want.

Tip for Mac OS users : You can find the path to ozc and ozengine by going to your application folder, "click with two fingers" on Mozart, chose the second item in the menu (something about seeing the content), the folder of the application will open and you will be able to find the bin folder somewhere containing all the oz binaries. By going to the properties of the ozc or ozengine, you will be able to get their full path and execute them by command lines.

6.2 Summary of what to do

The project has to be done in groups of two people, you cannot do it alone. No plagiarism will be tolerated (you can discuss ideas with each other but never share your code).

6.2.1 Mandatory part

Your submission has to contain all the files provided to you as a base. It is advised to reuse code between your pacmOz and ghOzt implementations when their behaviour is the same. For example, they could both use the same function to know which moves are available from their current position. These functions should be put in a separate file and imported in both agents.

6.2.2 Optional part

Succesfully implementing all of the above will give you a grade of 14/20. To get more points, you have to implement additional features amongst the following:

- Add a bonus that allows pacmOz to go through ghosts unharmed (like they are invisible). These bonuses should spawn on the map in certain locations.
- Create a Maze generator.
- Allow people to control pacmOz with their keyboard.
- Improve the graphical interface.
- Add custom sounds to the game.
- Develop an "AI" system for the pacmOZ and the ghOzts. We do not expect you to use advanced AI techniques, but you could make pacmOz go towards pacgums while avoiding the ghOzts for example, or make the ghOzts communicate together to trap pacmOz.

Of course, if you have a great idea that is not listed here, you can add it. In this case, make sure to detail it in your report. You will not get all the bonus points if the basic part of the project doesn't function properly. In order to obtain 20/20, you have to implement at least 2 of the six options listed above. It is obvious that changing the color of the walls for example is not enough to get bonus points for changing the interface you have to make a significant improvement.

6.3 Submission

Your submission will be done on INGInious. You have to submit your report as a pdf and a .zip file containing the different files of your project.

- Your `Main.oz` file containing the code of your Game Controller agent.
- At least one `PacmOzXXXname.oz` file (where XXX is your group number (ex: group 42 has to put 042), and name is the name of this PacmOz (ex: basic, random, ...) allowing to differentiate your different agent. If you have made an AI as an extension, please put that implementation in a separate file.
- At least one `GhOztXXXname.oz` file (where XXX is your group number (ex: group 42 has to put 042), and name is the name of this GhOzt (ex: basic, random, ...) allowing to differentiate your different agent. If you have made an AI as an extension, please put that implementation in a separate file.
- Your `Input.oz` file.
- Your `AgentManager.oz` file.
- A Makefile allowing us to compile your project.
- A folder named `ress` containing all the sprites used.
- A folder named `extension` that contains the extended version of the game if it not interropable.

Your report should be maximum 3 pages long, and detail your implementation details, the feedback you gathered from the interoperability, and the extensions you implemented. Please only give the relevant information. The deadline for the project is **Wednesday December 6th, at 23:59**.