Final competition

By MiMaCh System - course of Intelligent Systems a.y. 21/22

TABLE OF CONTENTS

Introduction

Description of the game

CodinGame Platform

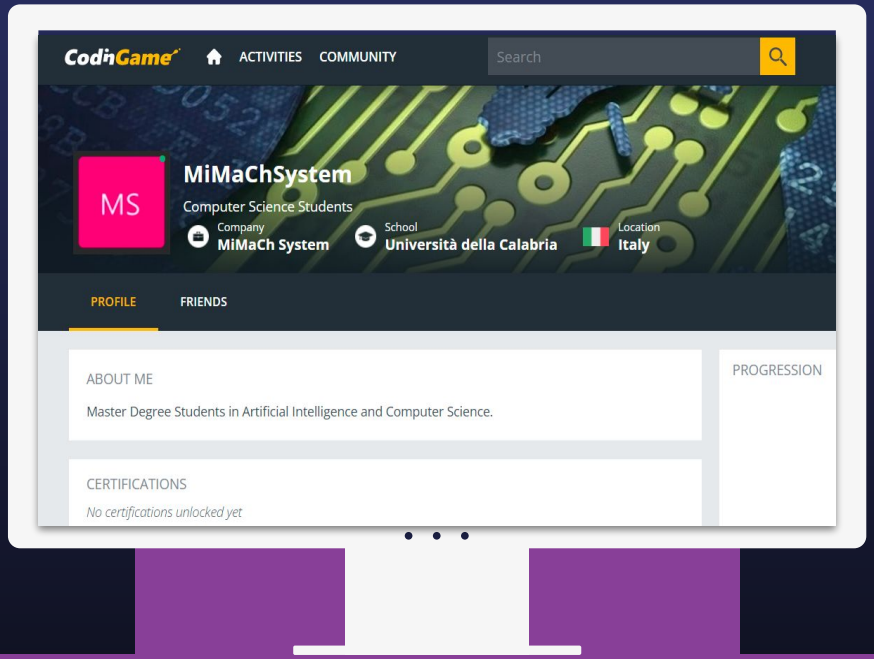
Brief introduction to the platform

Pacman with a twist

Brief description of the game

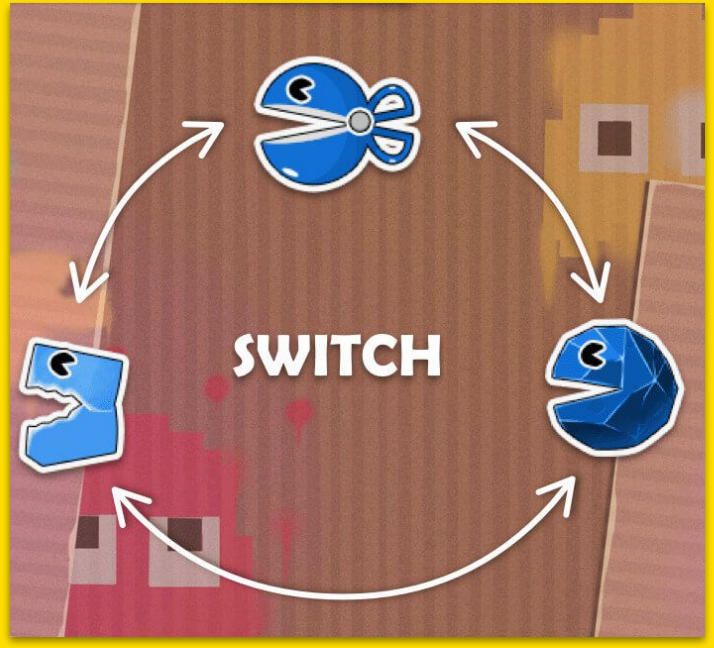
The Platform CodinGame

- On CodinGame, passionate developers learn, train, chat and have tons of fun!
- On the home page you can find a quick tutorial and introduction about the platform.
- The goal of this platform is to let programmers keep on improving their coding skills by solving the World's most challenging problems, learn new concepts, and get inspired



Spring Challenge 2020

- This Challenge is a **Pacman** game but with a nice twist: pacmans can transform themselves and eat each other following the rules of the game **rock-paper-scissors**
- It is really interesting and attracting version of the classical arcade game!
- This puzzle can be solved using different topics, also studied during our lectures:
  - Pathfinding
  - Multiagent
  - Distances & 2D Array
- It is possible to find more information on the website Practice AI, Pathfinding and Multi-agent with "Spring Challenge 2020"



Main Rules

- The game is played on a grid, composed by walls and floors.
- Each player controls a team of pacs.
- The map is filled with pellets (**1 point**) and super-pellets (**10 points**).
- The game stops when there are no enough pellets to change the final result or after 200 turns.

Target

- Eat more pellets than your opponent!
- Avoid getting killed!
- The winner is the player with the highest score, regardless of the amount of surviving pacs.

Description of the Environment

Partially Observable

Our pacmans have vision on pellets and enemies connected by a continuous straight line. Super-pellets can be seen from everywhere.

Deterministic (Strategic)

Each decision made by our agents determines next steps.

Sequential

Each decision depends on the previous ones and has effect on the next ones. Each turn is not episodic.

Dynamic

Even if the grid does not change and the pallets do not move, our actions and opponents actions change the condition of the play and pellets count.

Discrete

It is available a finite number of possible decisions, perceptions and actions.

Multiagent

We control a number of pacs between 2 and 5 against the same number of opponents.

Description of the Agent

- At each turn, we have information regarding the visible pacs and pellets.
- For each pac, we have its identifier, whether it's ours or not and its coordinates.
- Each pac has access to two abilities (**SWITCH** and **SPEED**) with the same cooldown period of 10 turns.
- Or it can **MOVE** to the target position

The approach adopted

1° step:
BRAINSTORMING

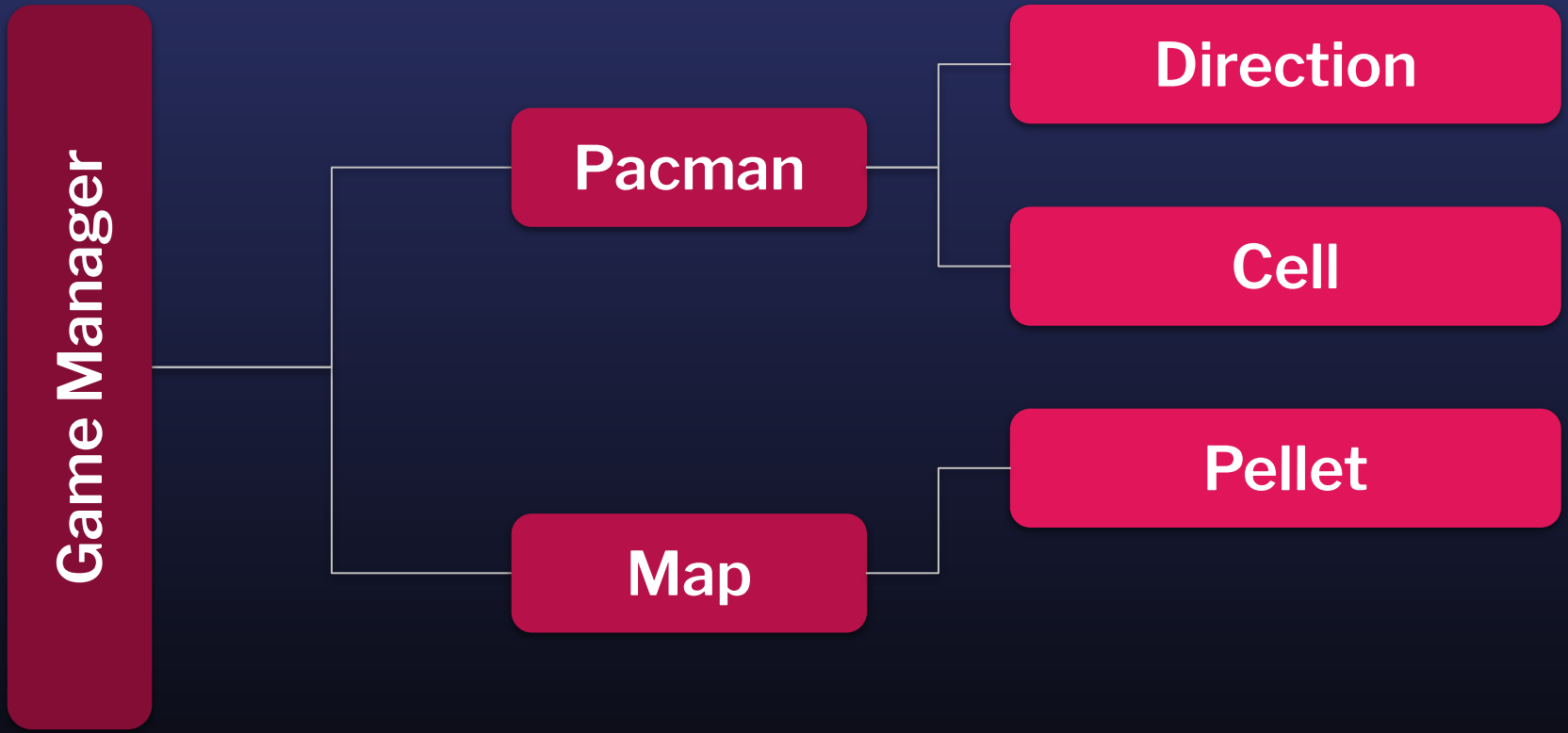
We started our approach by sharing ideas, main goals and possible strategies to apply.

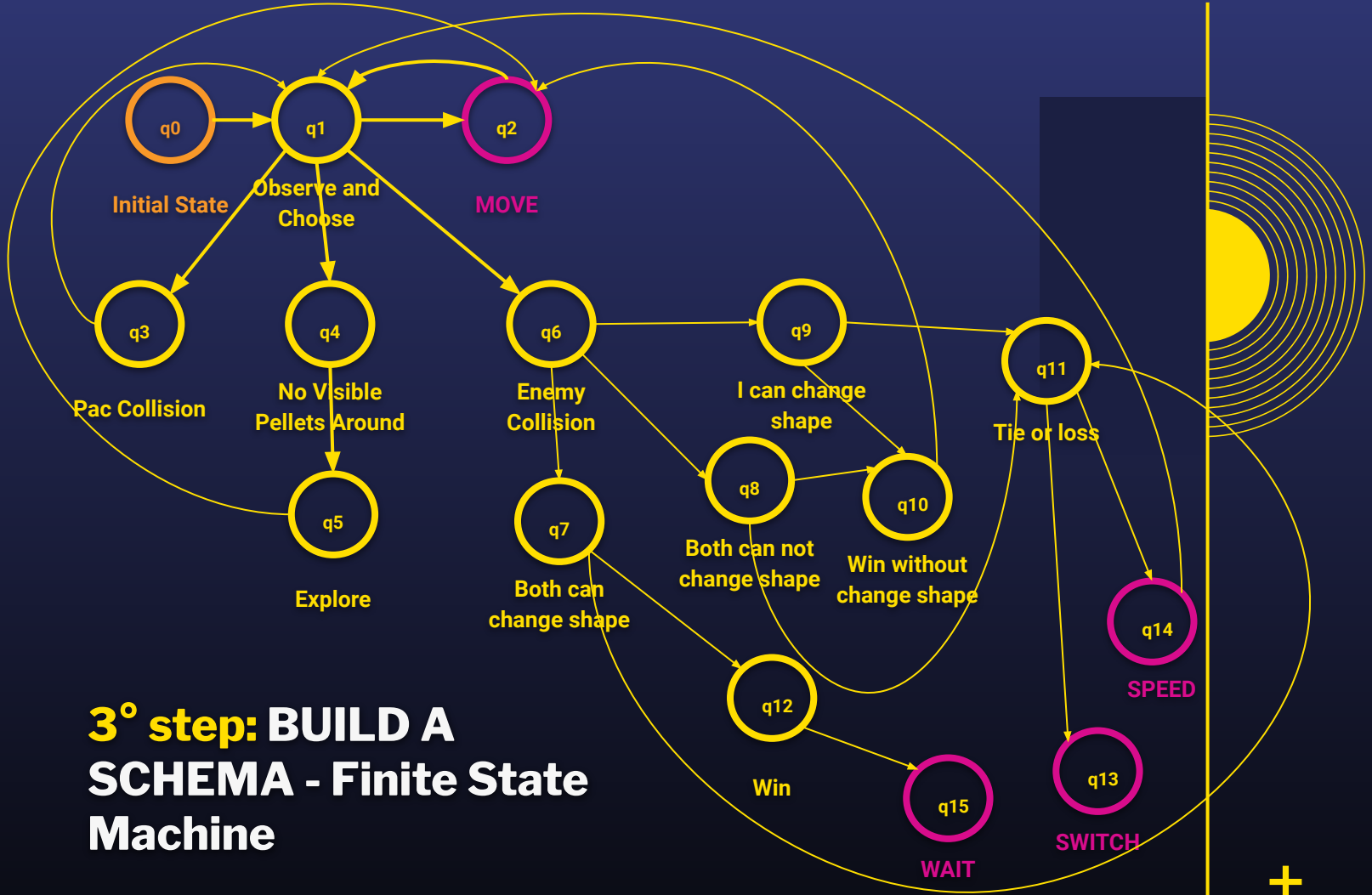
- **Analyze the game**
- **Analyze the possible situations**
- **Be specific**

We wrote everything down on paper, writing in natural language.

2° step: MODELING

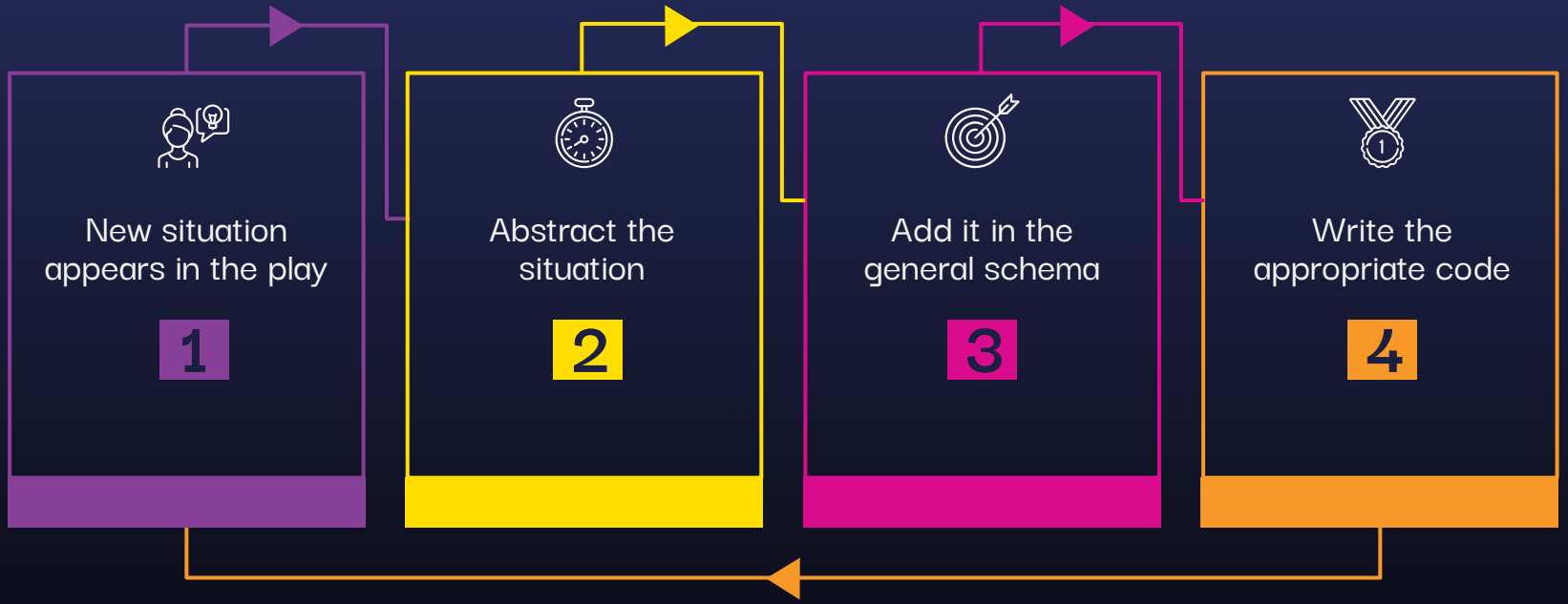
We modeled the World in a Diagram Class



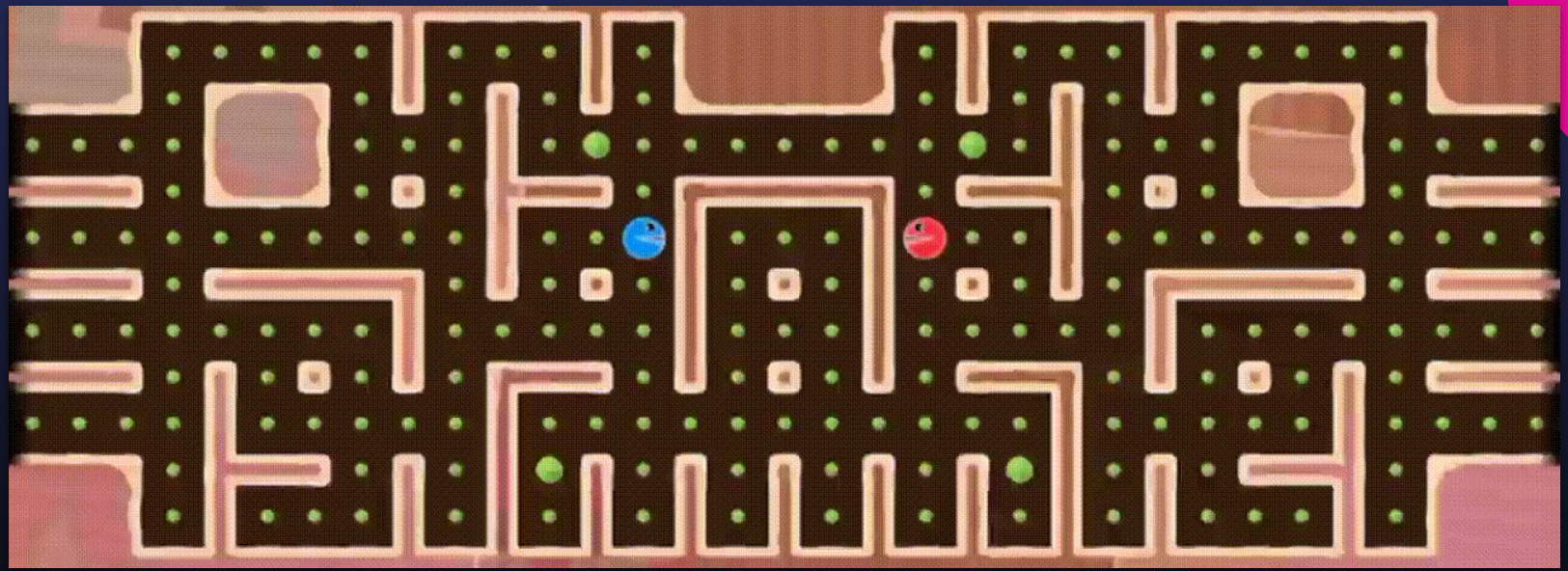


3° step: BUILD A SCHEMA - Finite State Machine

A CYCLIC APPROACH



Implementation



Game Manager

FIELDS

SETUP

```
class GameManager{
    Map board;
    ArrayList<Pacman> myPacmans;
    ArrayList<Pacman> opponents;

    ArrayList<Pacman> myLastPosition;
    ArrayList<Pacman> opponentsLastPosition;

    ArrayList<Pacman> superPellets;

    int myScore;
    int opponentScore;
    String output;

    public GameManager(int w, int h){···

    void stampaMappa(){···

    void setBoard(String line, int row){···

    void addMyPacman(Pacman p){···

    void addOppoPacman(Pacman p){···

    void setScores(int my, int oppo){···

    void setVisible(ArrayList<Pellet> visible){···

    void addvisible(Pellet p){···

    void clearAll(){···

    void clearPos(){···

    void updateMap(){···
```

UTILITY

```
boolean isSuperPellet(int x, int y){···

void chooseDirection(){···

void explore(Pacman p){···

int fightResult(Pacman my, Pacman oppo) {···

Pacman isOpponentNear(Pacman p) {···

void findPacman(Pellet p){···

void closestCell(ArrayList<Cell> cells, Pacman p){···

boolean presente(ArrayList<Cell> cell, Cell c){···

void goAway(Pacman p, Pacman near){···
```

MAINCORE

```
void checkCollision(){···

void checkForNull(){···

void checkForFight() {···

void checkForSuperPellets(){···

void activeSpeed(){···
```

```
void genOutput(){···

void play(){···
```

```
class Pacman{

    int pacId;
    int x;
    int y;
    String typeId;
    int speedTurnsLeft;
    int abilityCooldown;
    ArrayList<Direction> possiblesMoves;
    Direction choice;
    Cell explore;
    String action;
    String switchTo;
    boolean iChoose;
    boolean locked;

    public Pacman(int pacId, int x, int y, String typeId, int speedTurnsLeft, int abilityCooldown)

    void setLocked(boolean b)···

    boolean isNear(Pacman p, int i){···

    void bestDirection(){···

    void tryToWin(Pacman oppo) {···

    void changeDirection(String notDir){···
}
```

```
class Map{
    char[][] mappa;
    int width;
    int height;
    ArrayList<Pellet> visible;

    public Map(int w, int h){···

    void setVisible(ArrayList<Pellet> visiblePellet){

    void addVisible(Pellet p){···

    void setRow(String row, int i){···

    void stampa(){···

    boolean checkPellet(int x, int y){···
}
```

Pacman & Map

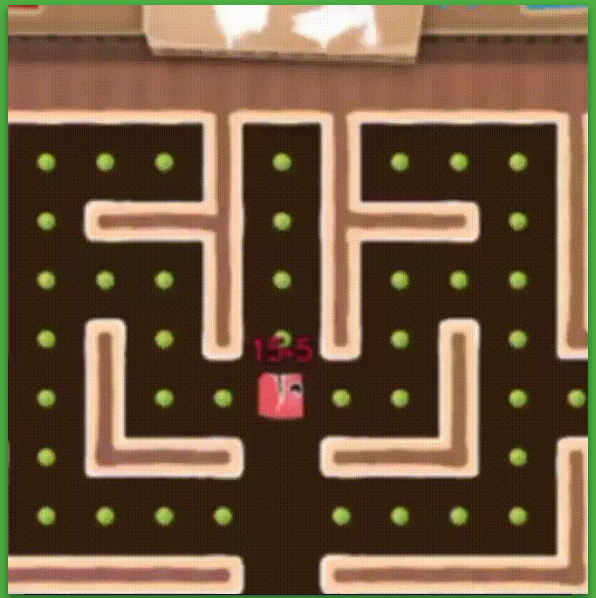
Cell, Direction & Pellet

```
class Cell{
    int x;
    int y;

    public Cell(){}

    public Cell(int x, int y){···

    public void setCoord(int x, int y){···
}

class Pellet{

    int x;
    int y;
    int value;

    public Pellet() {···

    public Pellet(int x, int y, int value){···
}
```

```
class Direction{
    String direction;
    double pointsOnTime;

    public Direction(String d, int points, int time){

    void setDirection(String d){···

    void setPointsOnTime(int points, int time){···

    void setDirection(Pacman my, Pacman op){···
}
```

Manage the movement

`MOVE pacId X Y`



- For the **Wood** leagues, we fixed a **target position** to our player. The target was the closest **SuperPellet** or the closest **Pellet**.
- The `distance(x,y)` method simply returns the distance between two points in a cartesian plane.

- For the **Bronze** and **Silver** leagues, we fixed a **direction** to our players. It can **move** or **explore** according to this direction.
- We chose a different approach because the player has a different perception of the environment. It can see only up and down its **y** and left and right its **x**.
- The player chooses its **best direction** by observing the possible directions and then it calculates the best according to the **ratio** between points over time.
- Otherwise, if there are no points around it, the player will explore an unvisited cell.

Manage New Skills

SWITCH

`SWITCH pacId pacType`

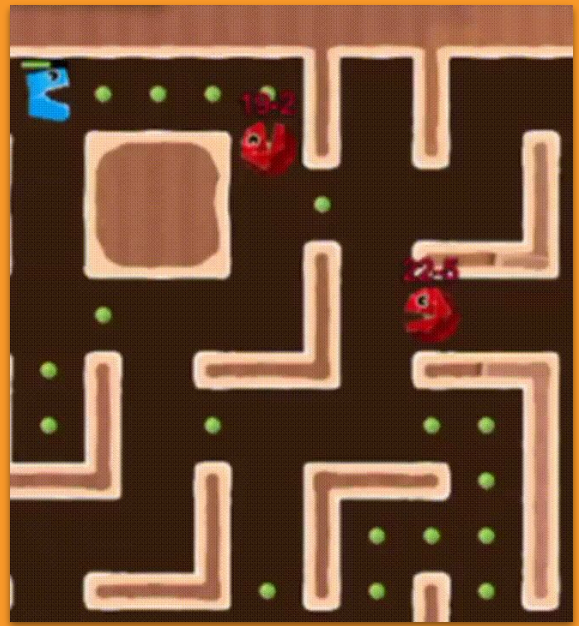
With pacType equal to:
- SCISSORS
- ROCK
- PAPER

(- DEAD in the Silver level)

SPEED

`SPEED pacId`

Allows a pac to move by 2 steps instead of 1 during the next 5 turns.



How to activate the SWITCH

I can switch

- If I win -> MOVE to the enemy
- If I lose -> SWITCH
- If it is a tie -> SWITCH

Both can switch

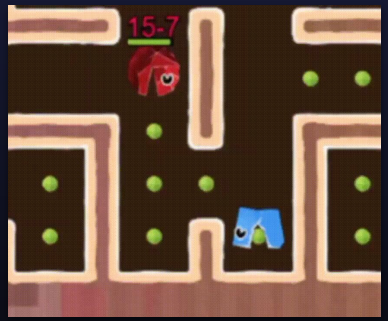
- If I win -> WAIT
- If I lose -> goAway()
- If it is a tie -> goAway()

Nobody can switch

- If I win -> MOVE to the enemy
- If I lose -> goAway()
- If it is a tie -> goAway()

Opponent can switch

- In any case -> goAway()




```
void checkForFight() {
    for(Pacman p: myPacmans) {
        Pacman near=isOpponentNear(p);
        if(near!=null) {
            if(p.abilityCooldown==0 && near.abilityCooldown!=0) {
                if(fightResult(p, near)==1){ //vittoria
                    p.action="MOVE";
                    if(p!=null && near!=null){
                        p.choice=null;
                        p.explore=new Cell(near.x, near.y);
                    }
                }
                else // sconfita o pareggio
                    p.tryToWin(near);
            }
            else if(p.abilityCooldown==0 && near.abilityCooldown==0) {
                if(fightResult(p, near)==1){ //vittoria
                    p.action="WAIT";
                }
                else //sconfita o pareggio
                    goAway(p, near);
            }
            else if(p.abilityCooldown!=0 && near.abilityCooldown!=0) {
                if(fightResult(p, near)==1){ //vittoria
                    p.action="MOVE";
                    p.choice=null;
                    p.explore=new Cell(near.x, near.y);
                }
                else //pareggio o sconfitta
                    goAway(p, near);
            }
            else if(p.abilityCooldown!=0 && near.abilityCooldown==0) {
                goAway(p, near);
            }
        }
    }
}
```

How to activate the SPEED

```
void checkForNull(){
    for(Pacman p: myPacmans){
        if(p.iChoose==false && p.action.equals("")){
            if(p.abilityCooldown==0)
                p.action="SPEED";
            else{
                explore(p);
            }
        }
    }
}
```

We decided to activate the SPEED ability when the pacman explores, in order to quickly get new points.

But, in the Silver League, we noticed that the Boss intelligence always actives the SPEED. So we adapted our strategy.

EXPLORE

```
void activeSpeed(){
    for(Pacman p: myPacmans){
        if(p.abilityCooldown==0){
            p.action="SPEED";
        }
    }
}
```

Silver League

How works the method

`explore(Pacman p)`

```
void explore(Pacman p){
    ArrayList<Cell> cell = new ArrayList<Cell>();
    ArrayList<Cell> visited = new ArrayList<Cell>();
    cell.add(new Cell(p.x, p.y));
    boolean found=false;
    while(found==false){
        ArrayList<Cell> temp = new ArrayList<Cell>();

        for(Cell c: cell){
            if(board.mappa[Math.floorMod(c.y-1, board.height)][c.x]!='#')
                temp.add(new Cell(c.x, Math.floorMod(c.y-1, board.height)));
            if(board.mappa[Math.floorMod(c.y+1, board.height)][c.x]!='#')
                temp.add(new Cell(c.x, Math.floorMod(c.y+1, board.height)));
            if(board.mappa[c.y][Math.floorMod(c.x+1, board.width)]!='#')
                temp.add(new Cell(Math.floorMod(c.x+1, board.width), c.y));
            if(board.mappa[c.y][Math.floorMod(c.x-1, board.width)]!='#')
                temp.add(new Cell(Math.floorMod(c.x-1, board.width), c.y));
        }
        visited.addAll(cell);
        cell=new ArrayList<Cell>();

        for(Cell po: temp){
            if(board.mappa[po.y][po.x]=='o' || board.mappa[po.y][po.x]=='O' || board.mappa[po.y][po.x]==' '){
                boolean ok=true;
                for(Pacman pac:myPacmans){
                    if(pac.pacId!=p.pacId && pac.explore!= null && (pac.explore.x==po.x && pac.explore.y==po.y))
                        ok=false;
                }
                if(ok){
                    p.explore=new Cell(po.x, po.y);
                    found=true;
                }
            }
            else if(presente(visited, po)==false)
                cell.add(po);
        }
    }
```

Check adjacent cells

Check if the cell to explore is already visited

```
ArrayList<Cell> celle=new ArrayList<Cell>();
if(p.explore!=null){
boolean up=true, down=true, right=true, left=true;
int i=1;
    while(up || down || right || left){
        if(up){
            int y=Math.floorMod(p.explore.y - i, board.height);
            if(board.mappa[y][p.explore.x]=='#'){
                up=false;
            }
            else if(board.mappa[y][p.explore.x+1]!='#' || board.mappa[y][p.explore.x-1]!='#'){
                celle.add(new Cell(p.explore.x, y));
            }
        }
        if(down){
            int y=Math.floorMod(p.explore.y + i, board.height);
            if(board.mappa[y][p.explore.x]=='#'){
                down=false;
            }
            else if(board.mappa[y][p.explore.x+1]!='#' || board.mappa[y][p.explore.x-1]!='#'){
                celle.add(new Cell(p.explore.x, y));
            }
        }
        if(right){
            int x=Math.floorMod(p.explore.x + i, board.width);
            if(board.mappa[p.explore.y][x]=='#'){
                right=false;
            }
            else if(board.mappa[p.explore.y+1][x]!='#' || board.mappa[p.explore.y-1][x]!='#'){
                celle.add(new Cell(x, p.explore.y));
            }
        }
        if(left){
            int x=Math.floorMod(p.explore.x - i, board.width);
            if(board.mappa[p.explore.y][x]=='#'){
                left=false;
            }
            else if(board.mappa[p.explore.y+1][x]!='#' || board.mappa[p.explore.y-1][x]!='#'){
                celle.add(new Cell(x, p.explore.y));
            }
        }

        i++;
    }
}
closestCell(celle, p);
p.action="MOVE";
```

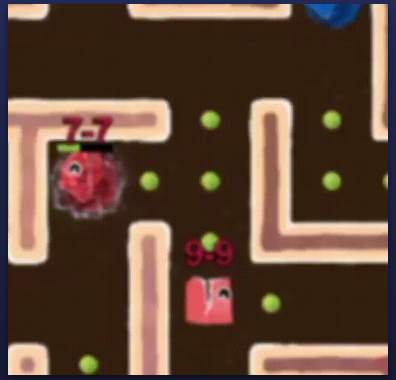
This second part check in an iterative way if there is a cell that allows us to see in advance the explorable cell

The strategy core

```
void play(int turn){
    clearPos();
    updateMap();
    chooseDirection();
    checkForNull();
    checkCollision();
    checkForFight();
    activeSpeed();
    genOutput(turn);
}
```

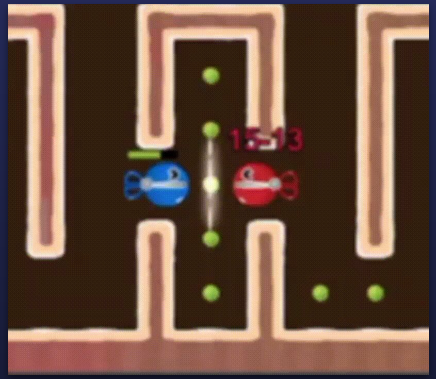
- The method `play()` is the main core of our strategy.
- It is the implementation of our finite state machine described previously.
- Thanks to this method it is possible to have a clear vision of our chosen strategy.

Manage Collision



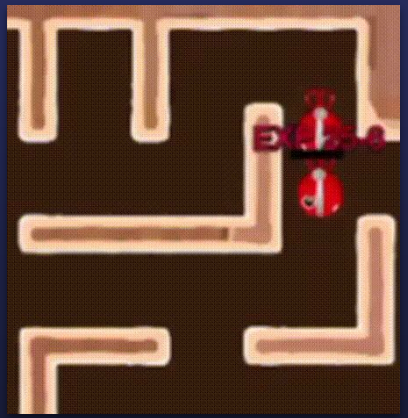
With our Allies

Otherwise both pacs will be locked in a loop.



With Enemies

With the same type.



While Exploring

Otherwise pacs can explore the same cells.

Strengths and Weakness

Collision Management

Switch strategy

The good world modeling

The exploration is still not optimized

Grids without tunnels represent a difficulty for our strategy

Super Pellets are not our target and often for then we do not reach enough score

Other Criticism

To our code:
Not high level of generalization

To the challenge environment:
By reaching the Silver League the Switch strategy is not more used because of the ability SPEED is broken

CONCLUSION - OUR RESULTS

Result

League

RESOURCES

Online

- Practice AI, Pathfinding and Multi-agent with "Spring Challenge 2020"
- CodinGame
- CodinGame Profile - MiMaCh System
- github.com - MiMaCh-System-Proj - final competition

Text Book

- Intelligenza artificiale. Un approccio moderno. Vol. 1 - Stuart J. Russell - Peter Norvig - - Libro - Pearson - | IBS

Thanks!

Do you have any questions?

CS@CS

UNIVERSITÀ DELLA CALABRIA

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**

MiMaCh System

- **Canonco Martina** [231874]
- **Morello Michele** [223953]
- **Passarelli Chiara** [223971]