Optimization of Pac-Man strategies using Model Predictive Control

Andrea Morghen , Antonio Gargiulo , Valentina Giannotti and Emmanuel Grimaldi

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, Università degli Studi di Napoli Federico II, via Claudio 21, 80125 Napoli, Italy

Identification and optimal control project, Professor Luigi Gliemo, 2023

Abstract— This paper provides a model based on optimal decision-making strategies for the pursuit game of Pac-Man. In our simplified version, Pac-Man is a game involving a pursuit-evasion dynamic with a single adversary who can randomly choose from four predefined patterns. In addition to evading the adversary, the agent must also pursue multiple stationary pills in an obstacle-filled environment.This work introduces an innovative approach that offers two implementable patterns to be chosen from. The agent can decide to get all the points or escape from the ghost. The results show that the artificial agent developed with this approach is able to evade the ghost and achieve victory in the game.

Keywords—PAC-MAN game, MATLAB, Traveling Salesman, Model Predictive Control

I. Introduction

The game of Pac-Man is one of the timeless classics in the field of video games, known for its combination of fast-paced action, strategic planning and maze solving. Pac-Man is a good example of multiple-objective games. The player (capable of commanding Pac-man) has the dual purpose of chasing pills and escaping from pursuers. The game takes place in a environment populated by obstacles. As such, it provides an excellent reference problem for a number of applications including optimal path choice, multi-objective search, constrained optimization.
In Pac-Man, each ghost implements a different decision policy, some with predefined patterns, some with random seed path. As a result, the game requires decisions to be made in real time, based on observations of a stochastic and dynamic environment that is challenging for both human and artificial players. Over the past decades, the desire to create an intelligent controller capable of beating the Pac-Man game has fueled much research in the field of artificial intelligence (AI) and control strategies.

The main goal of our work is to develop a simplified version of the Pac-Man game in MATLAB, and along with it a controller that is able to cope some of the game's complex and dynamic challenges. To achieve this, we will adopt an approach based on search algorithms and adaptive strategies. We will take advantage of the current state of the overall system to determine the optimal game strategy that enables the controller to make intelligent decisions in real time.

One of the key elements of our controller will be the use of search algorithms, such as the traveling salesman algorithm, used to select the most efficient route for Pac-Man to collect all the pills. Additionally, we will implement adaptive strategies that allow the controller to adapt to the ever-changing dynamics of the game, such as the possible aggressiveness of the ghost.

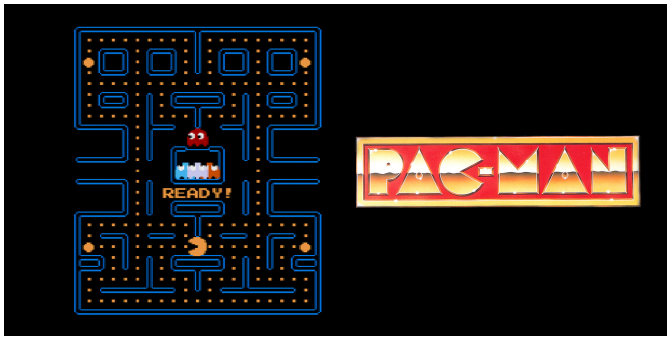

Fig. 1: Pac-Man game, 1980

Our work aims to contribute to the progress in the field of control applied to the Pac-Man game, providing a simple but structurally robust project, which can be a basis for those wishing to start approaching advanced control problems. The choice of the development environment, MATLAB, is a clear indicator of this, in fact MATLAB is the most used tool in the university environment. The results obtained may be used as a basis for the development of new game strategies, as well as provide insights for the application of search algorithms and adaptive strategies in other complex contexts.

II. PAC-MAN vs. MATLAB

In this section, we present our approach to creating a simplified model of the famous game PAC-MAN using MATLAB for the purpose of automatic control. The idea is to represent all the elements of the game's maze as elements of a matrix. These elements can be either static and immutable, such

as the perimeter walls and obstacles, or dynamic, including pills, PAC-MAN itself, and the ghost. Our simplified model features a basic maze represented by a 7x17 matrix, denoted as the 'maze' variable in the MATLAB code [5]. Additionally, there is a single ghost present in the maze. It is worth noting that our code is designed to be portable and can be adapted to any maze structure, regardless of size or the presence of different obstacles. While our model lacks certain elements found in the original game, such as the tunnel effect, that allows PAC-MAN to move from one end of the maze to the other, or the super pills, that grant power-ups to eat the ghost, these additions can be easily implemented for future developments.

a. PAC-MAN game environment

The game environment in our PAC-MAN model is fully represented by a matrix called "maze". Each element of this matrix assumes a specific value corresponding to the game element it represents. Here, we describe the various values and their meanings within the maze matrix:

- **maze(i,j)=0** represents a feasible maze tile without a pill. These tiles are still accessible to PAC-MAN, but they do not hold any additional incentives.
- **maze(i,j)=1** represents a wall (either a perimeter wall or a simple obstacle). These walls are static and cannot be modified throughout the game.
- **maze(i,j)=2** represents a feasible maze tile that contains a pill. These pills serve as points of interest for PAC-MAN and are an essential aspect of the game.
- **maze(i,j)=3** represents the position of PAC-MAN.
- **maze(i,j)=4** represents the position of the ghost.
- **maze(i,j)=7** represents the entrance to the ghost's home, this door remains static throughout the game.

Within our model, there is a unidirectional transition from a value of 2 to 0 when Pac-Man consumes a pill. In other words, a tile that initially contains a pill (value 2) loses the pill (value changes to 0) once it has been consumed by Pac-Man. The structure of the maze, represented by the maze matrix, enables us to visually depict the game's evolution using appropriate MATLAB graphical toolboxes, such as the Image Processing Toolbox [6].
Additionally, this matrix-based representation allows us to approach the various dynamic models of the game elements, such as the movement of PAC-MAN and the ghost, as well as their interactions with the maze walls, pills, and empty tiles. Below is the structure of the maze matrix and its visualization. Our analyses were based on that structure. As mentioned, these analyses continue to be valid on any other maze type.
The initial condition of the game, which is strictly dependent on the initial values of the **maze matrix**, is as follows:

$$\mathbf{maze} = \begin{bmatrix} 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 1&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&1 \\ 1&2&2&1&1&1&2&1&7&1&2&1&1&1&2&2&1 \\ 1&3&2&1&0&1&2&1&4&1&2&1&0&1&2&2&1 \\ 1&2&2&1&1&1&2&1&1&1&2&1&1&1&2&2&1 \\ 1&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&1 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \end{bmatrix}$$

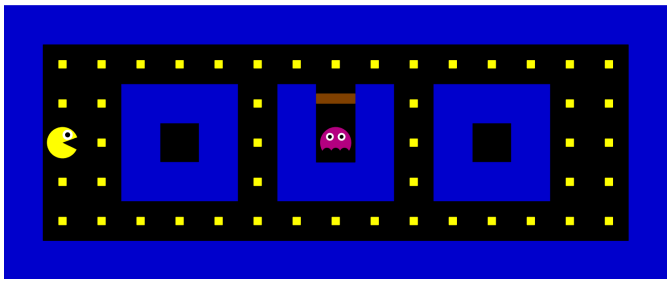

Fig. 2: Maze model of PAC-MAN in MATLAB

b. PAC-MAN

In our model, Pac-Man is the sole element represented by the value 3 within the maze matrix. The movement of Pac-Man within the maze is associated with the manipulation of the matrix indices that identify the unique element equal to 3. To capture Pac-Man's dynamics, we employ a discrete-time model due to the maze's two-dimensional nature. For Pac-Man's dynamic model, we consider two state equations, one along the y-axis (updating the row index) and the other along the x-axis (updating the column index). Each state equation operates in discrete time and has two inputs: one with a positive sign (incrementing the index) and one with a negative sign (decrementing the index).
In this matrix, at each time step, only one of the four inputs is active (satisfying the constraint $\sum(u) = 1$), indicating the direction of Pac-Man's movement. Consequently, we obtain a 2x2 dynamic matrix represented by an identity matrix and an input matrix which follows the above construction.

State vector of Pac-Man model:

$$\mathbf{z_p} = \begin{bmatrix} y_p \\ x_p \end{bmatrix}$$

Input vector of Pac-Man model:

$$\mathbf{u_p} = \begin{bmatrix} uy_{1p} \\ uy_{2p} \\ ux_{1p} \\ ux_{2p} \end{bmatrix}$$

Dynamic equations describing Pac-Man's movement:

$$y_p(k+1) = y_p(k) + uy_{1p} - uy_{2p}$$
$$x_p(k+1) = x_p(k) + ux_{1p} - ux_{2p}$$

It is possible to derive the matrices of the system with ease.

Dynamic matrix:

$$\mathbf{A_p} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Dynamic input matrix:

$$\mathbf{B_p} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

These matrices identify the model we will use as the MPC constraint.

The input vector, consisting of four elements, will be appropriately calculated by the implemented control algorithms based on the desired behavior and strategy for Pac-Man within the maze.

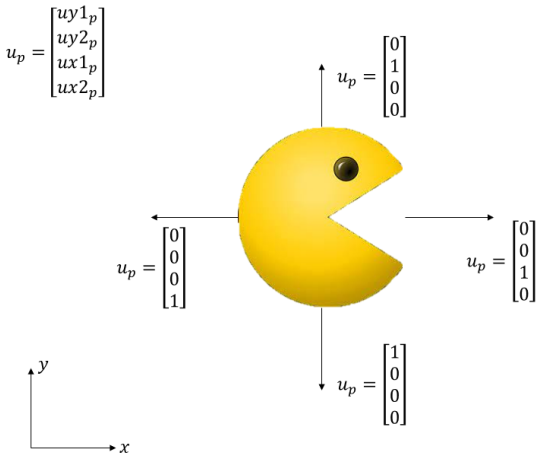

Fig. 3: PAC-MAN model

c. Ghost

Similar to Pac-Man, the ghost's movement is described by a dynamic model that operates on the two coordinates, updating the indices corresponding to the element 4 within the maze. However, there is a crucial difference in the ghost's dynamic model compared to Pac-Man's. Each state equation of the ghost's model has a single input, which is not binary but can take the values of -1, 0, or 1, one of the two inputs must be active (satisfying the constraint $\sum(|u|) = 1$). Where we respectively denote the decrease, null action, and increase of the associated index.

State vector of ghost model:

$$\mathbf{z_g} = \begin{bmatrix} y_g \\ x_g \end{bmatrix}$$

Input vector of ghost model:

$$\mathbf{u_g} = \begin{bmatrix} uy_g \\ ux_g \end{bmatrix}$$

Dynamic equations describing ghost's movement:

$$y_g(k+1) = y_g(k) + uy_g$$
$$x_g(k+1) = x_g(k) + ux_g$$

In the original version of the game, the ghosts follow a fixed pattern. However, in our example, such a pattern would be too simplistic. Therefore, we devised a different approach to simulate the ghost's movement. In a first approach we had thought of a random choice at each step but this returned an unnatural movement of the ghost. So we decided to formulate the ghost's movement as a random choice among four distinct paths, with the goal of reaching three fixed positions within the maze.
In other words, whenever the ghost arrives at one of the three fixed positions, it randomly selects a path (i.e., a sequence of inputs) from its model to reach the next fixed position. By implementing this randomized movement strategy, we achieve a dynamic and unpredictable ghost movement that changes with each game. This enhances the excitement and challenge for Pac-Man, as the ghost's actions are not predictable or mechanical.The implementation of this random movement strategy for the ghost adds an engaging and dynamic element to the game, keeping the interaction between Pac-Man and the ghost captivating and preventing repetitive patterns.

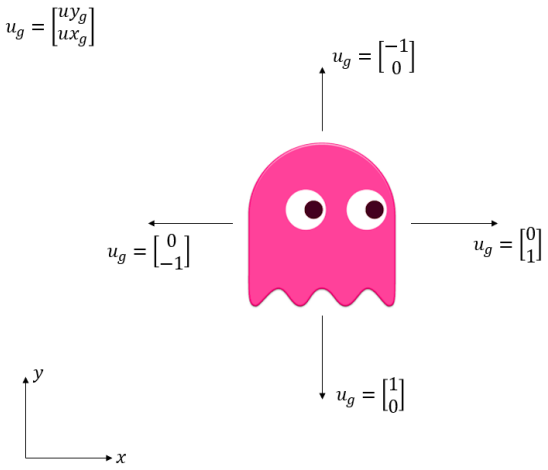

Fig. 4: Ghost model

d. Maze evolution

The evolution of the game is made possible by the dynamic models described above. Specifically, these models, following specific rules, allow for changes in the values of the maze matrix at each step. It is through the evolution of the maze matrix that the game scenario unfolds, and at each step, the game possibilities are evaluated, and the appropriate control actions are applied. In particular, when Pac-Man moves to a cell containing a pill, the value of that cell in the maze matrix changes once Pac-Man leaves it. The value changes from 2 (pill present) to 0 (pill consumed). However, if the cell already had a value of 0, there is no change. If the ghost moves into the cell occupied by Pac-Man, the game ends, and the element 3 representing Pac-Man disappears from the maze matrix. At each step, a check is performed to determine whether the game is over or not. In particular, it checks whether there are any cells left in the maze matrix with value 2. If there are no such cells, it means that Pac-Man has consumed all the pills and won the game and a win message is displayed. Alternatively, the presence of element 3 in the maze matrix is checked. If it is not present, it means that Pac-Man has been captured by the ghost and a game over message is displayed. The evolution of the maze matrix, driven by the dynamics of Pac-Man and the ghost, plays a crucial role in shaping the

game's progression. It enables the evaluation of game possibilities and the application of appropriate control actions at each step. In addition, this approach promotes visualization of the game dynamics.

III. PROBLEM FORMULATION

This section presents the decision-making strategy used by Pac-Man to make sure to win the game by meeting some optimality criteria. The artificial player will have to choose the direction in which to move by taking into account various objectives:

- Minimize the number of steps useful for capturing all pills currently present;
- Enact an escape policy the moment Pac-Man is below a critical distance from the ghost;

These two main objectives are achieved by using two control strategies based on solving the traveling salesman problem [4] and the optimization problem arising from the use of model predictive control.[2]

IV. CONTROL STRATEGIES

The control algorithm used to decide the input to be provided to Pac-Man among those available is different depending on the situation in which it is located. During the operation of the game, an iterative check is made on the distance between the positions of the ghost and Pac-Man by considering the Euclidean norm as a metric. The obtained value is compared with a threshold "critical distance" and from this comparison the control decides the function to use in order to calculate the optimal value of the control action. The "critical distance" is a control's degree of freedom of the designer because is useful to implement or not an aggressive escaping policy. For example, increasing it aims to achieve a more conservative control that prefers escape.

a. MPC

The Model Predictive Control (MPC) [1] algorithm is an advanced control approach used to make optimal decisions in real time, it is based on the formulation of an optimization problem that considers a finite time horizon and the Pac-Man dynamics.
In this context MPC is applied to determine the optimal directions for the Pac-Man agent to escape from the ghost and in the meantime minimize the distance from the nearest pill. In the simulations the finite horizon has a length N=3 and the controller applies only the first element of the control sequence that resolve this optimization problem:

$$\begin{aligned} &\max \sum_{k=0}^{N-1} \alpha \, ||\mathbf{d}||_1 - \beta \, ||\mathbf{f}||_1 \\ s.t. \quad &\mathbf{z_p}(k+1) = A\mathbf{z_p}(k) + Bu(k) \\ &\mathbf{1}^T\mathbf{u} = 1 \\ &y_p(k+1) \geq up_b \\ &y_p(k+1) \leq lw_b \\ &x_p(k+1) \geq lf_b \\ &x_p(k+1) \leq rg_b \\ &100 * y_p(k+1) + x_p(k+1) - obs(i) + 0.5 \leq 1000 * w(k)(i) \\ &-100 * y_p(k+1) - x_p(k+1) + obs(i) + 0.5 \leq 1000 * w(k)(i + length(obs) \\ &w(k)(i) + w(k)(i + length(obs)) = 1 \end{aligned} \tag{1}$$

$$i \in [1 : length(obs)] \quad k \in [1; N]$$

Where:

- α: Weight variables associated with distance from the ghost
- β: weight variables associated with distance from the nearest pill
- $u(k)$: binary input variable
- $\mathbf{d} = (\mathbf{z_p} - \mathbf{z_g})$: difference between Pac-Man and ghost's coordinates;
- $\mathbf{f} = (\mathbf{z_p} - \mathbf{n})$: difference between Pac-Man and nearest pill coordinates, calculated using a simple minimum distance algorithm;
- up_b= 2 upper bound
- lw_b= 6 lower bound
- lf_b= 2 left bound
- rg_b= 16 right bound
- obs= vector containing encrypted obstacle locations
- $w(k)$= binary auxiliary variables

The optimization problem thus defined is easy to interpret with regard to the first six constraints, which guarantee the evolution of the state variables according to the LTI system (A,B), the mutual exclusivity of inputs (only one active) and spatial limits linked to the extremes of the game map. The last three constraints are more interesting and worthy of a more in-depth analysis. These constraints prevent Pac-Man from choosing a next state with an obstacle. To do this, all the pairs (y,x) of points in the maze matrix were recovered, to which an obstacle corresponded (maze(y,x)==1 | maze==7). The coordinates of these points have been encrypted and saved in the obs vector according to the following logic:

$$obs(i) = 100 * y + x$$

This logic allows a transformation from 2 variables to 1 variable without loss of information. With the help of binary support variables, a Boolean negation policy has been implemented. The constraint

$$100 * y_p(k+1) + x_p(k+1) - obs(i) + 0.5 \leq 1000 * w(k)(i)$$

requires the auxiliary variable w(k)(i) to be positive if the predicted position y(k+1),x(k+1), encrypted with the same logic used previously, is greater than or equal to the encrypted value of the obstacle. In a similar way the constraint

$$-100 * y_p(k+1) - x_p(k+1) + obs(i) + 0.5 \leq 1000 * w(k)(i + length(obs$$

requires the auxiliary variable w(k)(i+length(obs)) to be positive if the predicted position y(k+1),x(k+1), encrypted with the same logic used previously, is less than or equal to the encrypted value of the obstacle. Consequently, the two variables w(k)(s) and w(k)(s+length(s)) will both be equal to 1 only and only if y(k+1),x(k+1) is exactly equal to the position of the obstacle. In order to prevent this from happening, the last constraint was imposed

$$w(k)(i) + w(k)(i + length(obs)) = 1$$

which ensures that such behavior never occurs, that is, for any future prediction and any present obstacle, the Pac-Man variables will never overlap with those of the obstacle.

The objective function is defined to minimize a cost that takes into account the distance between the Pac-Man agent and the ghost, while simultaneously trying to maximize proximity to the nearest pills. The MPC controller is developed using the Yalmip optimization toolbox in a Matlab environment [3]. The direction of movement is selected based on control input values represent North, South, East, West. In this way, the Pac-Man agent adopts the optimal movement strategy to avoid the ghost. After each iteration of the MPC, the distance between ghost and Pac-Man will be evaluated again to see if there is a need to reapply an escape or chase pill policy.

b. Salesman problem

The traveling salesman algorithm is a widely used method for solving combinatorial optimization problems in which a salesman must visit a set of cities while minimizing the total distance traveled. [4] This algorithm has been used to determine the optimal path that Pac-Man (salesperson) must follow to collect all the pills (cities). The classical approach involves generating all possible permutations of cities and calculating the total distance for each permutation.
In Matlab [5], the traveling salesman problem has been implemented in a dedicated file and called as a function. That function takes as input the maze(x,y) matrix containing the current state of the Pac-Man game and solves a constrained linear programming problem. The following steps are performed:

1. All possible trips, that is, all distinct pairs of stops, were generated, each is associated with a decision variable z_{ki} . To accomplish this, all coordinate pairs (x,y) that corresponded to either pills (value 2) or Pac-Man (value 3) in the maze(x,y) matrix were taken. After that, all weight coefficients c_{ki} are calculated using the Euclidean norm.

$$c_{ki} = \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2}$$

2. The distance for each possible trip was calculated. To ensure that the tour includes all stops, it is necessary to include the linear constraint that each stop is present in exactly two trips. This means one arrival and one departure from the stop.

$$\sum_{k=0}^{N} z_{ki} = 1 \ , \ \sum_{i=0}^{N} z_{ki} = 1 \qquad k \neq i$$

3. The cost function, sum of the chosen paths, was minimized. The decision variables z_{ki} are binary and associated with each trip, where each 1 represents an existing trip on the tour and each 0 represents a trip not on the tour.

$$\min \sum_{k=0}^{N} \sum_{i=0}^{N} z_{ki}\, c_{ki} \qquad k \neq i$$

4. Minimization presents a result with several subpaths. The constraints specified so far do not prevent these subtours from occurring. To prevent all possible subtours from occurring would require an incredibly large number of inequality constraints. Since it is not possible to add all subtour constraints, an iterative approach was taken. Once the subtours are identified in the current solution, inequality constraints are added to prevent these specific subtours from occurring, and the constrained LP minimization problem is re-solved. This approach allows for finding a suitable tour within a few iterations.

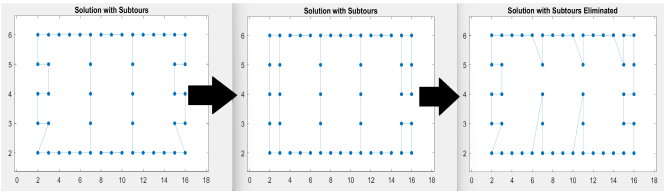

Fig. 5: subsequent approximations of the tour

5. Trips corresponding to the active decision variables are taken and reordered. The first trip will have Pac-Man's location as its start, all subsequent trips are selected to ensure that the end point of one trip matches the start point of the next trip.

The end result is a vector containing all trips to be performed to minimize the space traveled. This vector is provided as an output to the function. In the system model, the individual trip (i-th element of the vector) is taken and the correct control action calculated.

V. SIMULATION RESULTS

The matlab implementation based on the theoretical concepts discussed above allowed us to perform appropriate simulations. The large number of tests we performed consistently reported successes using the following weights $\alpha = 0.22$, $\beta = 0.11$ that demonstrates the efficiency of the control strategy. Excerpts from the simulations are given below. In particular, an MPC control case study with feasible directions highlighted is given.
Reported here is the case of context/control change, from MPC to traveling salesman, in which case the traveling salesman problem is solved with the current system data and the new solution calculated.

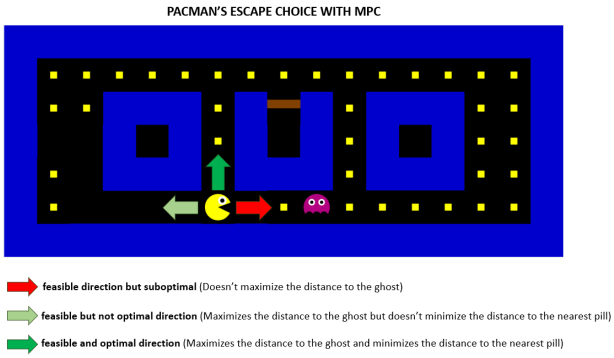

Fig. 6: Pac-man's escape choice with MPC

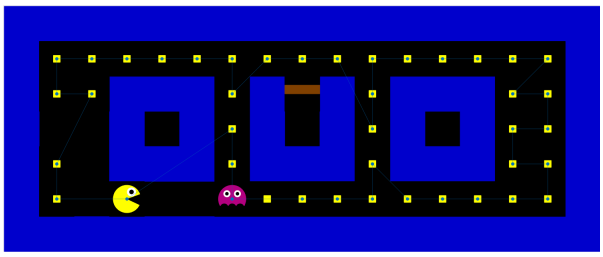

Fig. 7: Run-time creation of the connected graph for the traveling salesman

VI. CONCLUSIONS

In this paper, we looked at the application of a controller for the Pac-Man game solution. Through the use of control algorithms, we were able to develop a system that could achieve remarkable performance in the simplified version of the game. The goal of creating a primordial version of the system and controller, on which future developments or simple school studies are possible, has been achieved.
It is important to note that our work is not limited only to the game of Pac-Man, but can be extended to other similar games or control problems in wider contexts. The control techniques used can be applied to a wide range of scenarios, contributing to the development of intelligent and autonomous solutions.
In conclusion, through the use of a controller based on MPC techniques and selection algorithms, we managed to get an effective solution for the game of Pac-Man. We are confident that our work can be useful to those who want to tackle more advanced control problems for the first time.

REFERENCES

[1] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems. Cambridge University Press, 2017.*

[2] Luigi Glielmo, *An Introduction to Optimization*, Università di Napoli Federico II, lecture slides. Mar. 2023.

[3] *https://yalmip.github.io/example/standardmpc/*

[4] Mathworks, *Traveling Salesman Problem: Solver-Based* URL: https://it.mathworks.com/help/optim/ug/travelling-salesman-problem.html

[5] Andrea Morghen, Antonio Gargiulo, Valentina Giannotti, Emmanuel Grimaldi. PAC-MAN vs. MATLAB. *https://github.com/ValentinaGiannotti/Pac-Man-project.git*, Università di Napoli Federico II, May. 2023.

[6] ImageProcessingToolbox *https://it.mathworks.com/help/images/*