COMPSYS726 Assignment 1: Mario Expert System

Student: Adam Lee
380248615

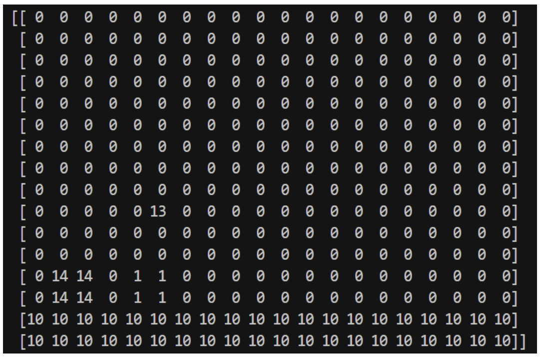
Table of Contents

1 Introduction

In this assignment, we were tasked to develop an expert system to beat Super Mario Land from GameBoy using Python. Therefore, three extra functions have been implemented into the expert system to be able to compute and beat the first stage of Mario. Current limitations are that it cannot avoid enemies behind Mario (For example if a Goomba is walking back towards Mario), and the limited number of entities Mario can handle, therefore limiting the amount of stages the expert system can compete.

2 Game Logic

The game is laid out in a 2D array and can be represented by numbers. For instance, Mario is represented by a 2x2 array of ones, the pipes are represented by 14 and the item blocks as 13. Each value of this array can be accessed, with the origin (0,0) starting from the top left corner.



```
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 13  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 14 14  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 14 14  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10]
 [10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10]]
```

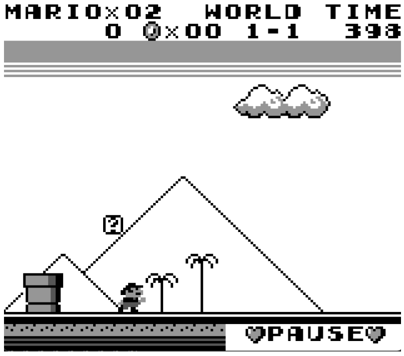


Figure 2.1: Array representation (Left) against visual representation of Mario (Right)

3 Expert System

The chosen expert system uses ***rule based inference*** to make decisions. This was chosen for the scope of this assignment as the first stage only has a few obstacles that Mario needs to avoid, is relatively simple to develop and modify, and the rule based inference is good for an unchanging environment (the enemies and the layout of a stage do not change each time a specific stage is played) (Grosan et al, 2011). The expert system features two key functions that independently scan Mario's x and y axes to detect potential threats like goombas and flies. The y-axis function, *entityRespondY*, takes priority over the x-axis function, *entityRespondX*, because it must react to threats above or below Mario, such as goombas and holes. An *overRideFlag* boolean in *entityRespondY* allows it to override actions in *entityRespondX*; otherwise, actions favour the x-axis response. Both functions use a *for loop* to scan each pixel ahead of Mario, applying the rule-based inference system to determine the appropriate actions.

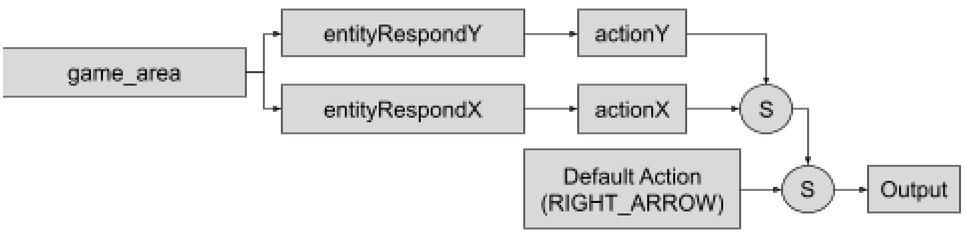


Figure 3.1: Block diagram of the hierarchy of actions

4 Addressing Challenges

4.1 Passing through multiple actions

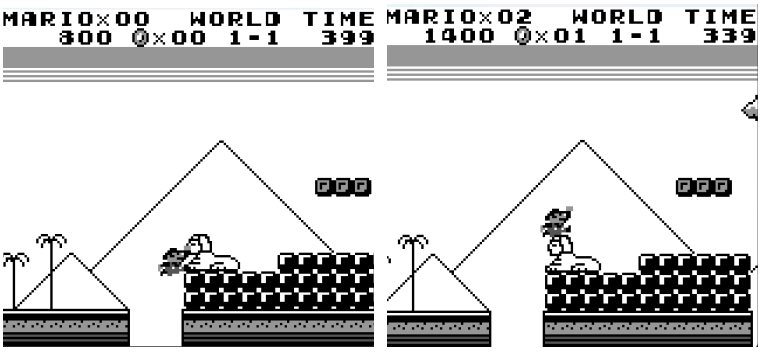


Figure 4.1: Comparison of Mario jumping (Left) against Mario jumping and moving right (Right)

When playing the Mario game, it is illogical as a human user to just press one button at a time. For instance, only pressing the button “A” when Mario approaches a hole will break his momentum and will result in him falling into that hole, shown in the screenshot here. Therefore, this was overcome by modifying the function “run_action” to take multiple inputs instead of single inputs, such that Mario can move right whilst jumping in the air, making him jump further.

```
    if isinstance(actions, int):
        actions = [actions]

    # Press each button in the actions list
    for action in actions:
        self.pyboy.send_input(self.valid_actions[action])

    # Hold the buttons for the specified duration
    for _ in range(self.act_freq):
        self.pyboy.tick()

    # Release each button in the actions list
    for action in actions:
        self.pyboy.send_input(self.release_button[action])

    self.prevActions = actions
```

Figure 4.2: Snippet of code displaying multiple actions being passed through

The code shown above illustrates how multiple actions are executed. The input parameter “actions” was modified in the original code such that “actions” is converted into a list if it isn’t one. Because of this, the “actions” parameter will have size varying between one or two depending on the scenario interpreted by the scanners of Mario. The exact same logic is done where for loops were implemented on the pressing and releasing of buttons to ensure all actions have been executed.

As a result, the jump shown 40 seconds into the video submission shows Mario successfully jumping over the hole and onto the sphinx platform.

4.2 Jumping in quick succession

Also, there is a bug in the game where pressing the button “A” (Jump) repeatedly in quick succession results in the jump action not being performed. Therefore, to overcome this, a conditional statement has been implemented into the actions selection function, where if the same action has been performed previously for the past 5 steps, it will assume that Mario is stuck. Consequently, the “*UP_ARROW”* button will be pressed, to break Mario’s stuck state. With this implemented, Mario will not get stuck in these scenarios.

This solution can be shown 15 seconds into the video when Mario runs against the wall for a short period of time before realising it is stuck and breaking that state. The same can be said 43 seconds into the video when Mario is stuck on the steps leading to the exit of stage one.

```
        if self.environment.prevActions == action:
            self.counter+=1

        elif self.environment.prevActions != action:
            self.counter = 0

        if self.counter == 5:
            self.counter = 0
            return MarioExpert.actions.UP_ARROW.value
```

Figure 4.3: Snippet of code displaying the lines of code to fix the jump bug

4.3 Implementing Timers

A pseudo timer integrated into the expert system can serve as a temporary interrupt, allowing Mario to perform a different action for a specified duration, effectively pausing the default rule-based behaviour. This interrupt mechanism is valuable because it enables the system to override the rule based logic when specific conditions arise. This is demonstrated 11 seconds into the video submission when there are two Goombas dropping above Mario. With the pseudo timer acting as an interrupt, if a Goomba is above Mario, the system will move him to the left to let the Goombas fall to the ground. However, the rule-based system is programmed to make Mario jump immediately when the Goomba is on the ground. Therefore in this case, the pseudo timer ensures that Mario continues moving left to maintain a safer distance from the Goomba before returning to the default rule based actions. By temporarily suspending the default action ("RIGHT_ARROW") and executing an alternative action, the system provides a safer and more robust response.

```
if game_area[i][marioX+3] == MarioExpert.icons.GOOMBA.value:
      self.initialStep = self.stepCount
      actionY = (MarioExpert.actions.LEFT_ARROW.value,2)
      return actionY, True
```

Figure 4.4: Snippet of code displaying where the pseudo timer is used

The pseudo timer was implemented by using stepCount as a unit of measurement (counting the amount of times the step function has been called). This pseudo timer can be used to delay certain actions based on the number of steps taken since the previous action, effectively simulating time-based behaviour without needing a real-time clock. For example if initialStep - stepCount is less than 10, it will ignore all rule based inference actions.

5 Quantifying Performance

Currently, the expert system is built towards beating the first stage only, and also as quickly as possible. Therefore Mario's primary objective will not be to collect all coins and power ups, but rather to beat the first stage as fast as possible. This is to maximise the score as each second is equivalent to 10 points. As seen in the video, the first stage can be completed in 72 seconds. This means that at the end of stage one, Mario gets an extra 3280 points from the time itself, totalling to 4980 points. In terms of object and entity avoidance, Mario does an excellent job, confidently beating the first stage without losing any lives, and the robustness of this code can be seen in the submitted video as Mario advances more than halfway through stage 2 before dying, accumulating 9680 points before the game is over. Collectively through stages 1 and 2, Mario has collected 36 coins. The statistics of the score is seen below.


```
{"lives": 0, "score": 9680, "coins": 36, "stage": 2, "world": 1,
"x_position": 1035, "time": 378, "dead_timer": 0, "dead_jump_timer": 0,
"game_over": true}
```


Figure 5.1: Final Score of the Mario Game

6 Next Steps and Improvements to the Expert System

Because the scope of the assignment is limited to beating stage one only, the expert system is not finalised to be able to complete all stages in Super Mario Land. For instance, the current expert system does not account for all the 25 entities, bosses, powerups, items and objects. Therefore, developing an extensive list of rules for the rule based inference system can become tedious and difficult (Grosan, et al 2011), and may introduce delays inside the scanning functions, preventing the code from running in real time, which is highly undesirable for this context.

Therefore, if the expert system is to be expanded further to beat more stages of the game, another approach to the inference method for the expert system must be considered such as fuzzy inference (Buckley et al, 1986), which can address uncertain scenarios and produce nuanced decisions based on different degrees of truth. To improve the expert system's to be more dynamic and realistic, developing the system to collect and use power ups, such as superballs, would greatly improve the gameplay. Having Mario reach the top exit at the end of the stage to play the bonus game is an additional feature.

7 Conclusion

Therefore, for the scope of this assignment only, the expert system developed for beating the first stage of Super Mario Land integrates rule-based inference. By implementing additional modifications to the code such as multiple action capabilities, addressing jumping issues, and incorporating a pseudo timer, the system achieves a robust performance, completing the stage in 72 seconds and accumulating a relatively good score. However, the system's current limitations include its inability to handle enemies behind Mario and the need for further development to address all game entities and stages. To do so, a different approach to the inference method for the expert system must be considered as the programming will be tedious. Future improvements could focus on incorporating power-ups, and expanding the logic to tackle additional stages, thus refining the system's adaptability and performance in a changing gameplay environment.

8 References

Grosan C., Abraham A. (2011) Intelligent Systems: A Modern Approach
https://link.springer.com/chapter/10.1007/978-3-642-21004-4_7

Buckley J.J., Siler W., Tucker D., (1986) A fuzzy expert system, Fuzzy Sets and Systems, Volume 20, Issue 1, 1-16.
https://www.sciencedirect.com/science/article/pii/S0165011486800276