ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – TP HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN

CSC14003 – AI PROJECT

SEARCH IN PACMAN GAME

GROUP MEMBERS

Trần Minh Đức	18127027
Nguyễn Vũ Thu Hiền	18127004
Ngô Thanh Phương Thái	18127208

LECTURER

Mr. Le Ngoc Thanh

Mrs. Ho Thi Thanh Tuyen

Thành phố Hồ Chí Minh, 16 Tháng Tám 2020

MỤC LỤC

I. INTRODUCTION

1. ABSTRACT

This project is aimed at implementing search algorithm on pacman problem that is able to find optimal path to eat coin at level 1, 2 and eating all coin without colliding ghosts at level 3, 4.

- 4 levels of game:

Level 1: Pac-man know the food's position in map and monsters do not appear in map. There is only one food in the map. Project can be separated with 3 subtasks.

Level 2: monsters stand in the place ever (never move around). If Pac-man pass through the monster or vice versa, game is over. There is still one food in the map and Pac-man know its position.

Level 2: monsters stand in the place ever (never move around). If Pac-man pass through the monster or vice versa, game is over. There is still one food in the map and Pac-man know its position.

Level 2: monsters stand in the place ever (never move around). If Pac-man pass through the monster or vice versa, game is over. There is still one food in the map and Pac-man know its position.

Our groups can separate requirements in 3 subtasks:

- Base game framework to simulate result of implemented algorithm
- Use basic search algorithm (BFS/DFS/UCS/A*) to use in Level 1, 2
- Use adversarial search in Level 3, 4

2. ENVIROMENT

- Language: Python 3.7+
- Graphic Game Framework: Pygame
- Enviroment Software: Anaconda
- Addition library: pytweening (optional)
- OS Build: Window 10 2004 (64 bits)

3. ASSIGNMENT PLAN

Milestone	Tasks	Date	Assigned	Progress

			to	
Base Game Structure	- Base game structure - Graphic display - Guild, rule to develop	3/8/2020 - 6/8/2020	Đức	100%
	- Map parser, sample search implements base on map	6/8/2020-7/8/2020	Đức	100%
Implement problem	- Design actions, problem interface for 2 subtasks	7/8/2020-8/8/2020	Đức	100%
	- Implement search on level 1	8/8/2020-9/8/2020	Hiền	100%
	- Implement search on level 2	8/8/2020-9/8/2020	Tuấn	100%
	- Implement search on level 3	8/8/2020-9/8/2020	Thái	100%
	- Implement search on level 4	8/8/2020-9/8/2020	Đức	100%
Optimization	- Finished game screen	9/8/2020	Hiền, Tuấn, Thái, Đức	100%
	- Optimize Alpha-beta pruning and Minimax on Adversarial Search	9/8/2020-12/8/2020	Đức	100%
Test, Build Map, Report	- Test, Build Map, Report	13/8/2020	Hiền, Tuấn, Thái, Đức	100%
Other features	- Menu, Group Info Menu, Exit	14/8/2020	Đức	80% (not friendly for ui)
	- Sound	14/8/2020	Hiền	100%

II. APPROACH SOLVE PROBLEM

- The maxium size of map supporting in project is 40x24 (30x30 pixel per 1 unit – large maps)
- The space of nodes is 960 nodes since our group choose BFS as search strategy of level 1 and 2. The number of nodes is small to caculate, and we don't necessary to implement A*, GBFS, UCS to make it more complexity
- Problem is pacman problem with ghost agents, wall, pacman agent, coin
- Action of pacman agent and ghost agents is defined as **0, 1, 2, 3** in **level 1, 2** and **"UP", "DOWN", "RIGHT", "LEFT"** in **level 3, 4** for easy debugging

1. LEVEL 1

1.1. Describe solution:

- The map is known fully so the solution of level 1 is find shortest path to food position. And the cost each step is -1 so we don't need to use UCS or any other algorithms using heuristic value.
- The space of problems is 960 nodes. Init state is pacman, goal is coin (food) position.

1.2. Pesudo code:

Breadth-first search on a graph

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier) /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

11

1.3. Implement:

1. *FirstGameScreen()* function

- Display first level screen

- Inherited from *PlayGameScreen()* in screens folder
- Including *on_key_down()* is an event handler function on the screen

```python
class FirstGameScreen(GameScreen):
    def __init__(self, state):
        GameScreen.__init__(self, state=state)
        self.titleFont = pygame.font.Font(PATH_ASSETS + "font/BD_Cartoon_Shout.ttf", 72)
        self.itemFont = pygame.font.Font(PATH_ASSETS + "font/BD_Cartoon_Shout.ttf", 48)

        print("Created [play first level screen]")
        self.tile_manager = FirstLevelManager()

    def on_key_down(self, event):
        if event.key == pygame.K_p:
            self.tile_manager.start()
        if event.key == pygame.K_LEFT:
            self.tile_manager.move_player(dx=-1)
        if event.key == pygame.K_RIGHT:
            self.tile_manager.move_player(dx=1)
        if event.key == pygame.K_UP:
            self.tile_manager.move_player(dy=-1)
        if event.key == pygame.K_DOWN:
            self.tile_manager.move_player(dy=1)

    def update(self):
        pass

    def render(self, window):
        window.fill((0, 0, 0))
        self.tile_manager.render(window)

    def clear(self):
        pass
```

2. FirstLevelManager()

- Inherited TileManager() in cores folder
- In additions, it implements function display result screen including “GAME OVER” and scores that user archived when the game ends
- Variable *finished == True* → end game
    - *game_over* → state of the game when ends
    - *score* → stored the scores that user archived

```python
if self.finished == True:
    pygame.display.set_mode((GAME_SETTING.M_WIDTH, GAME_SETTING.M_HEIGHT))
    pygame.display.set_caption(GAME_SETTING.TITLE)
    pygame.display.set_icon(pygame.image.load(GAME_ICON))

    game_over = self.titleFont.render("GAME OVER", True, (100, 0, 0))
    surface.blit(game_over, (70, 170))

    score = self.itemFont.render("Score: " + str(self.step), True, (100, 0, 0))
    surface.blit(score, (200, 275))
```

2. LEVEL 2

2.1. Describe solution:

- The map is still known fully and ghost stand in the place ever (never move around). The is one food in the map
- Start state is position of pacman, goal state is position of food
- Ghosts don't move since we treat ghosts as walls to solve this problem
- Algorithm is similarity to level 1 (use BFS)

2.2. Implement:

- In **"cores/agent/maze_problem"** defines not in ['1','3'] isn't able to collide wall or ghost

```python
def actions(self, s: MazeState):
        ms = s
        list_actions = []
        if ms.x > 0 and self.graph[ms.y][ms.x - 1] != '1' and self.graph[ms.y][ms.x - 1] != '3':
            list_actions.append(Action(0))
        if ms.y > 0 and self.graph[ms.y - 1][ms.x] not in ['1', '3']:
            list_actions.append(Action(1))
        if ms.x < 40 - 1 and self.graph[ms.y][ms.x + 1] not in ['1', '3']:
            list_actions.append(Action(2))
        if ms.y < 24 - 1 and self.graph[ms.y + 1][ms.x] not in ['1', '3']:
            list_actions.append(Action(3))
        return list_actions
```

- Screen: ThirdGameScreen **(similar with level 1)**

- Manager: ThirdLevelManager **(similar with level 1)**

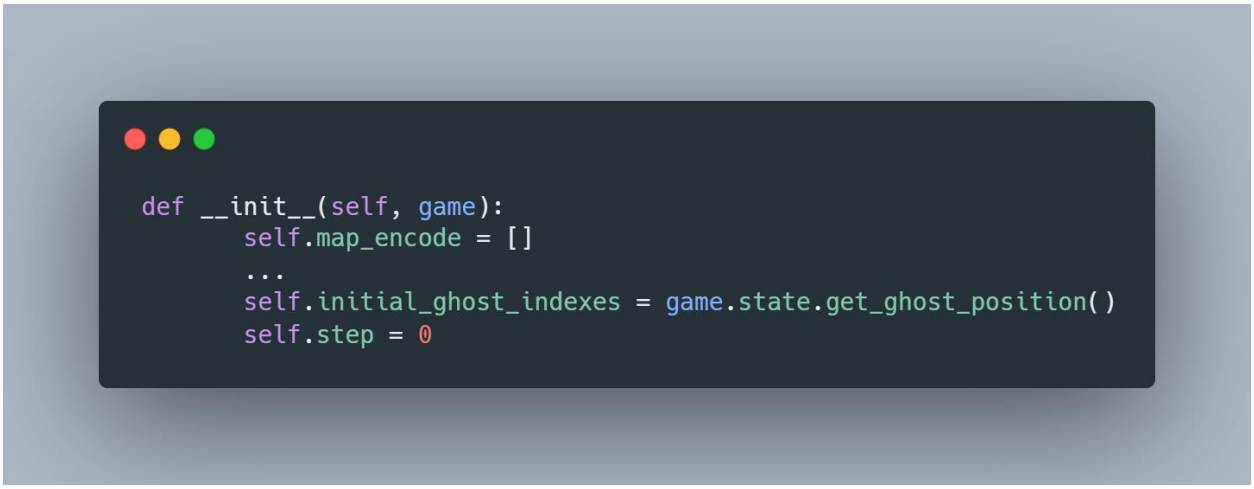
3. LEVEL 3

3.1. Describe solution:

- At level 3, pacman cannot see the foods if outside three-step. Monster just move one step. Each step pacman go each step monsters move.
- In theory with heuristic strategy, we must choose the effective heuristic to sovle this problem. In this case, we have to eat food and avoid ghosts since if we use inform search we need to solve multiple problems (corner problems, ghosts problem, and food problem).
- The solution of our group use in both level 3, 4 is adverserial search (Minimax and AlphaBeta Pruning)
- Wedevelop level 3 base on Minimax and Alphabeta Pruning algorithm at level 4. The detail of algorithm is described at “Level 4”

3.2. Implement

- The solve of problem is based on level 4, since At this level, we just need to make ghost move in one step base initial state.
  + Firstly, save first position of all ghosts at **ThirdLevelManager**



+ Check action which ghosts can move in all possible action. (Only accept 1 step)

```
    def update(self):
        game = self.game
        num_agents = len(game.agents)
        if self.finished == True:
            print("Finished ")
            return
        if game.game_over:
            print("End")
            self.finished = True
            return
        for agent_index in range(num_agents):
            agent = game.agents[agent_index]

            action = None
            observation = game.state.deepcopy()
            if agent_index == 0:
                observation.optimize_mahattan_danger()

            action = agent.get_action(observation)
            game.move_history.append((agent_index, action))
            temp_state = game.state.generate_successor(agent_index, action)
            if self.monster_can_move(agent_index, temp_state.get_ghost_pos(agent_index)) or agent_index
== 0:
                game.state = game.state.generate_successor(agent_index, action)
                game.rules.process(game.state, self)
                print("Agent {0} action: {1}".format(agent_index, action))
                print("Epoch agent {0}, Num moves{1}".format(
                    agent_index, game.num_moves))

                if agent_index == 0:
                    self.move_pacman(game.state.get_pacman_position())
                elif agent_index > 0:
                    self.move_monster(game.state.get_ghost_state(
                        agent_index).getPosition())
```

+ If pacman collide ghosts, change game_over is True

```
if game.state.collide_ghosts_pos(game.state.get_pacman_position()) == True:
                        game.game_over = True
                        print("Chet roi!!!!")
```

+ If pacman eats all foods, fishish game and return score screen

```
if game.state.is_win():
    self.finished = True
    print("An het coin roi")
```

+ Render score screen:

```
if self.finished == True:
    pygame.display.set_mode(
        (GAME_SETTING.M_WIDTH, GAME_SETTING.M_HEIGHT))
    pygame.display.set_caption(GAME_SETTING.TITLE)
    pygame.display.set_icon(pygame.image.load(GAME_ICON))

    game_over = self.titleFont.render(
        "FINISHED", True, (100, 0, 0))
    surface.blit(game_over, (70, 170))

    score = self.itemFont.render(
        "Score: " + str(self.game.state.get_score()), True, (100, 0, 0))
    step = self.itemFont.render(
        "Step: " + str(self.step), True, (100, 0, 0))
    surface.blit(score, (200, 275))
    surface.blit(step, (200, 350))
```

4. LEVEL 4

4.1. Describe solution:

- Level 4 map is opened. Monsters will seek and kill pacman. Pacman want to get food as much as possible. Pacman can see 3 nearest-step.
- With the explanation of level 3, we will use Adversarial search at this level to maximize pacman score and minimize ability killing of monsters.
- Monsters cannot go back suddenty (move 180 degree)
- The core of structure to implement this problem is base on an idea of simulation pacman project at **Berkeley University (effectively structure)**

4.2. Pesudo code:

- Minimax

The minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← -∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

21

- Alphabeta pruning

The alpha-beta search algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state,-∞,+∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state,α,β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← -∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a),α,β))
    if v ≥ β then return v
    α ← MAX(α, v)
return v
```

31

The alpha-beta search algorithm

```
function MIN-VALUE(state,α,β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) ,α,β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

4.3. Implement:

- All implementation of 2 algorithms are in “core/minimax” (both minimax and alpha-beta pruning)
- Depth of algorithm is 3, since pacman can see 3 nearest-step:

```
class MiniMaxAgent:
    depth = 3

    def __init__(self, depth=3):
        self.depth = int(depth)
```

- Function **def get_action(self, game_state: GameState):** to get the action with maximize score, if problem have many actions with same score, it will choose randomly action in those cases.
- The evaluation function is the current score of that GameState

```
def score_evaluation_func(current_game_state: GameState):
    return current_game_state.get_score()
```

+ if state collide ghost, the score – 1000

+ if state win game, the score + 1000

+ All step pacman moves will – 1 score

+ Each food pacman eats, score + 20

- Alpha-beta pruning is similar with minimax. The detail of two algorithms is **"cores/minimax/MiniMaxAgent"**

4.4. Optimization:

- We have any variables to caculate space of generate successors:

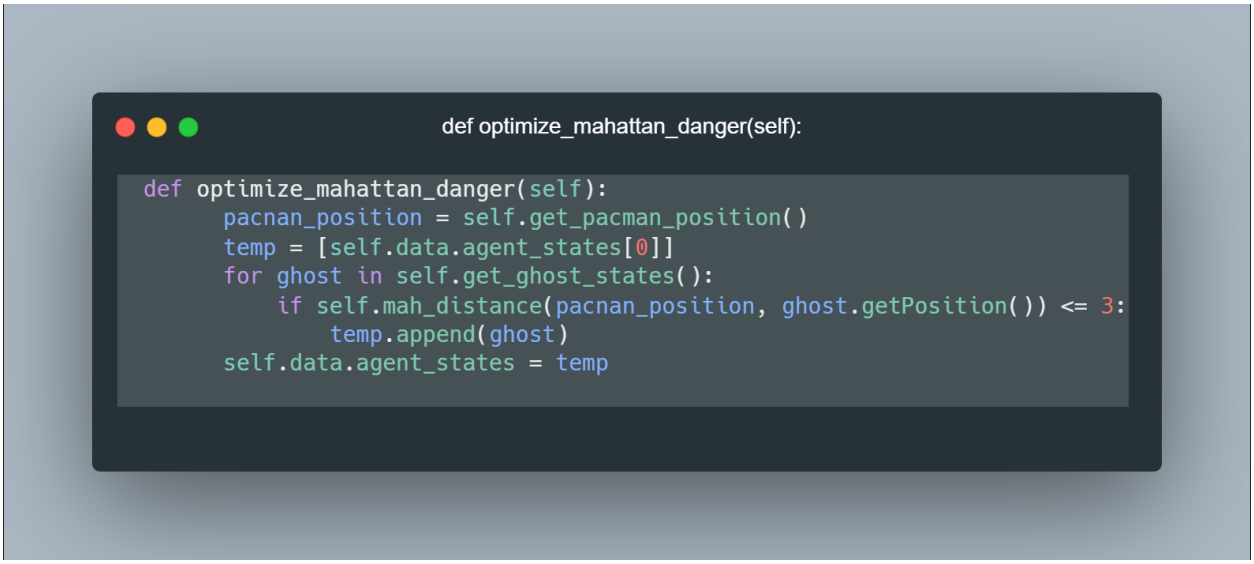
  + the number of ghosts: N(ghosts) = 2 (2 for example)

  + the number of actions: N(directions) = 4 (UP LEFT RIGHT DOWN)

- The space of a step pacman caculating is: (2+1) * 4= 12

  + Pacman can see 3 step so space is 12^3= 1728 conditions **(!! Very slow)**

- So we can decrease the space is check mahattance distance of pacman and ghost <= 3 step (the region can affect pacman suvival).
  + The best case we only need 4 conditions **(!!!! Interesting)**
- In **"cores/minimax/GameState.py"**



- And we call it to decrease the space to caculate Minimax or AlphaBeta Pruning Algorithm

```
for agent_index in range(num_agents):
    agent = game.agents[agent_index]

    action = None
    observation = game.state.deepcopy()
    if agent_index == 0:
        observation.optimize_mahattan_danger()

    action = agent.get_action(observation)
    game.move_history.append((agent_index, action))
    game.state = game.state.generate_successor(agent_index, action)
    game.rules.process(game.state, self)
    print("Agent {0} action: {1}".format(agent_index, action))
    print("Epoch agent {0}, Num moves{1}".format(
        agent_index, game.num_moves))
```

III. GAME STRUCTURE

- src

- assets: The resources of this project
- cores: core algorithm to search

  + agent: using for describing the problem in level 1, 2

  + minimax: the minimax and alphabeta search

  + search: bfs implementation

  direction.py: 4 directions

  FirstLevelManager.py: manage tile and algorithm in FirstGameScreen

  SecondLevelManager.py

  ThirdLevelManager.py

  FourthLevelManager.py

  TileManager.py: sample implementations of level 1, 2

  MinimaxManger: sample implementations of level 3, 4

- layers:
  entity: contain object to render use pygame library (Coin, Ground, Monster, Player, Wall)
- maps:

map_lv1.txt: map use for level 1

map_lv2.txt: map use for level 2

mini.txt: map use for level 3 + 4

- screens: contains all screens objet inherited from GameScreen. It contains event handler, polymorphism rendering. Easy to develop other screens
- states: contains state of game (not playing).

pacman.py: main game

gpath.py: global path in project include

setting.py: setting config for game

IV. GUILD TO USE

- Open workspace at folder “src”
- Run “python pacman.py”
- Choose level to run
  + If you choose level 1 or 2, we need to press P to start runing simulation (Since we solve the map first)
  + If you choose level 3 or 4, it will run automatically
- Change map at “maps”

  + “map_lv1.txt” for level 1

  + “map_lv2.txt” for level 2

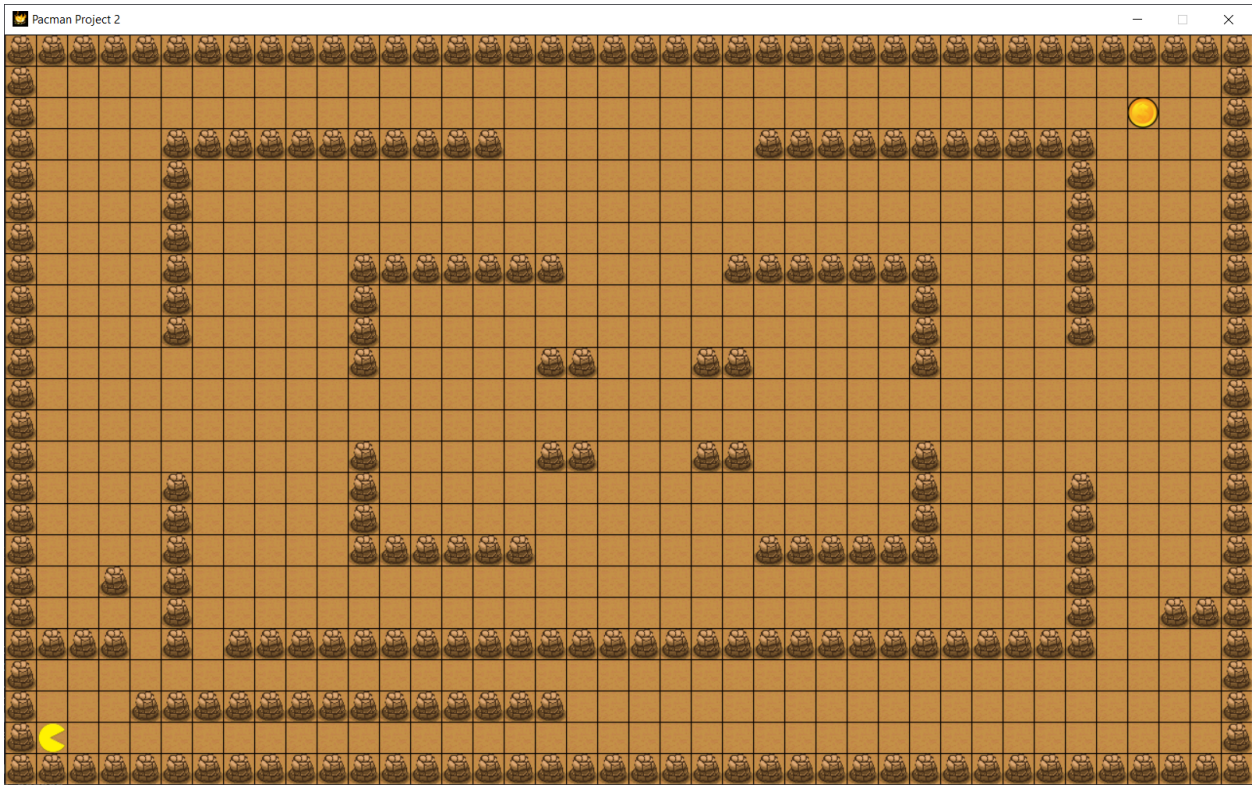
  + “mini.txt” for level 3 or 4

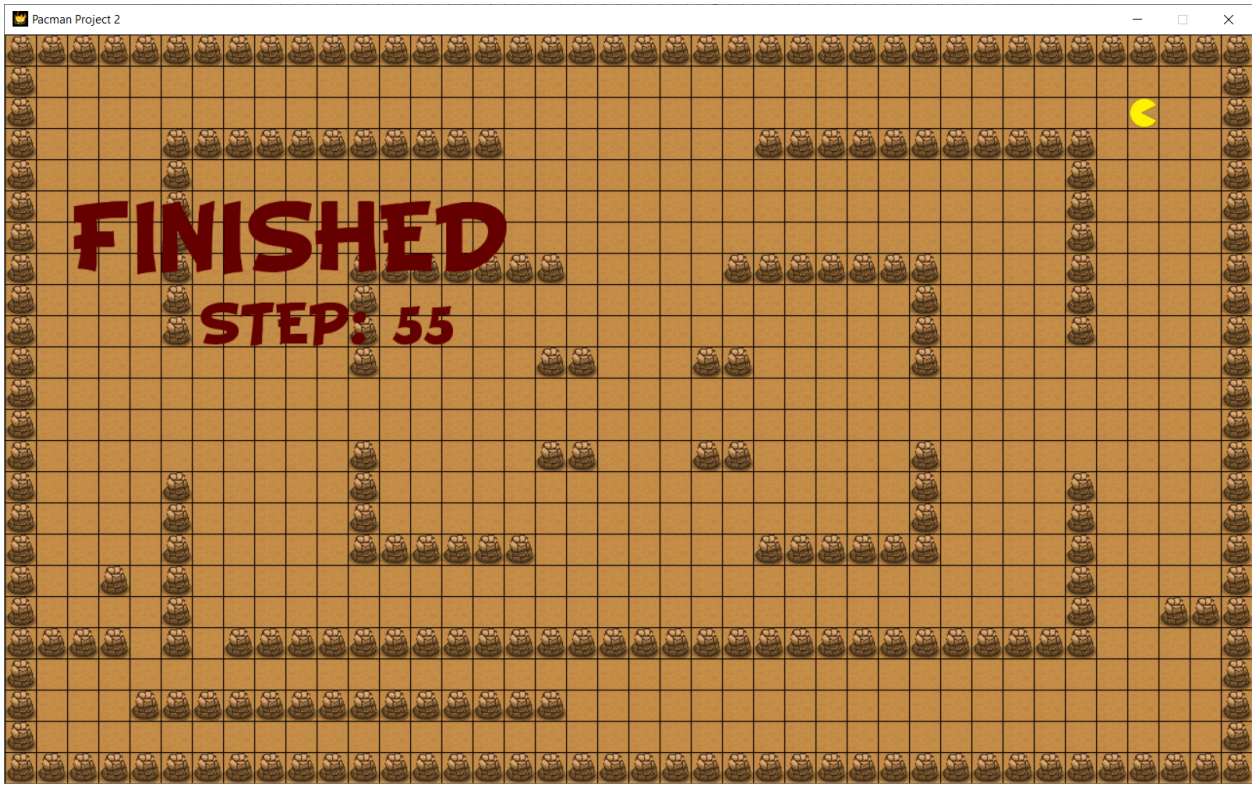
- If you want to change map in file for tesing, please close and reopen the game.

V. RESULT AND MEASURE

1. Level 1

- Map1:

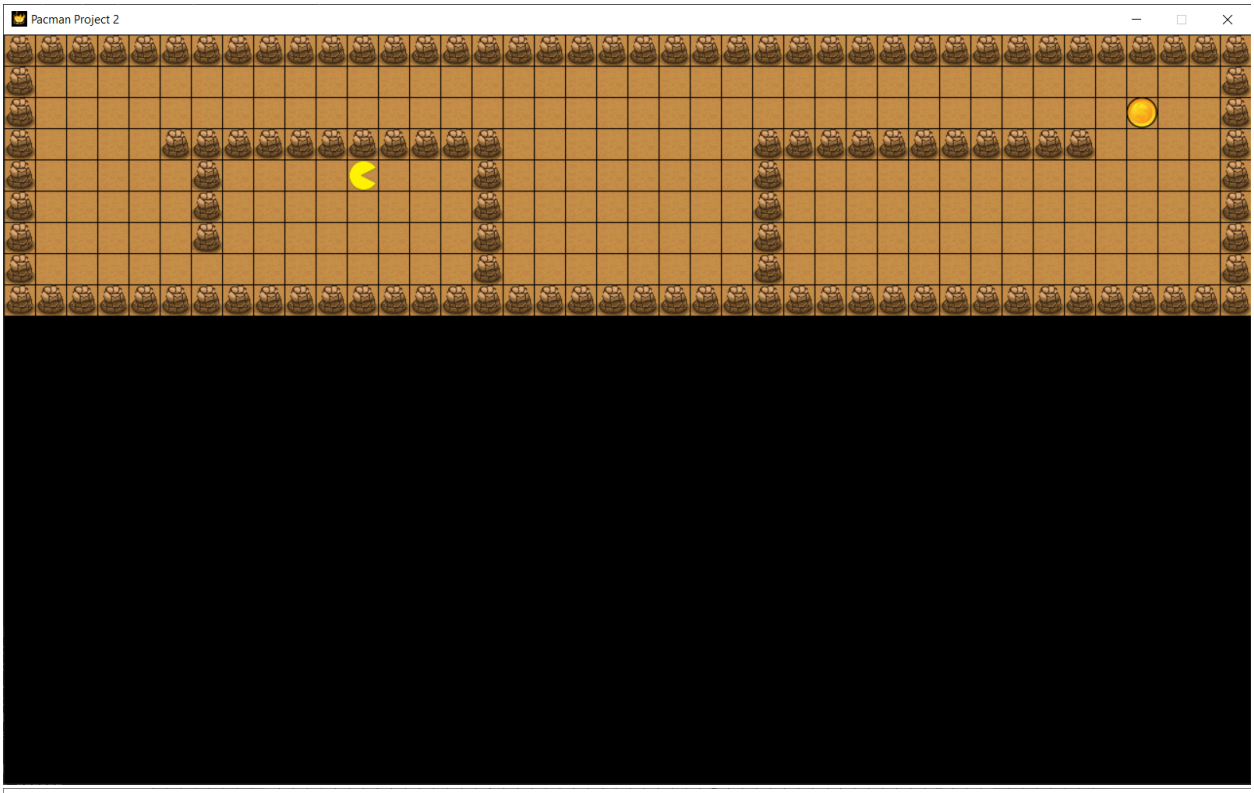


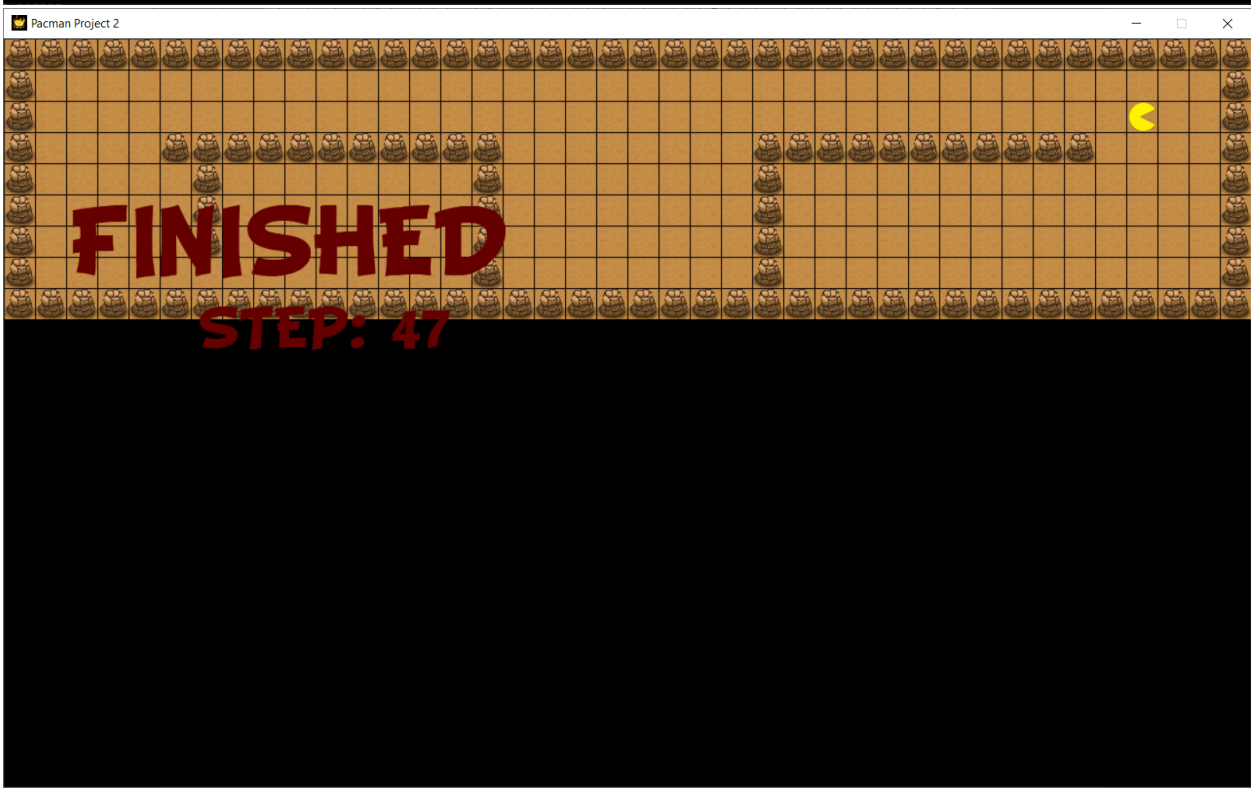


- Map2

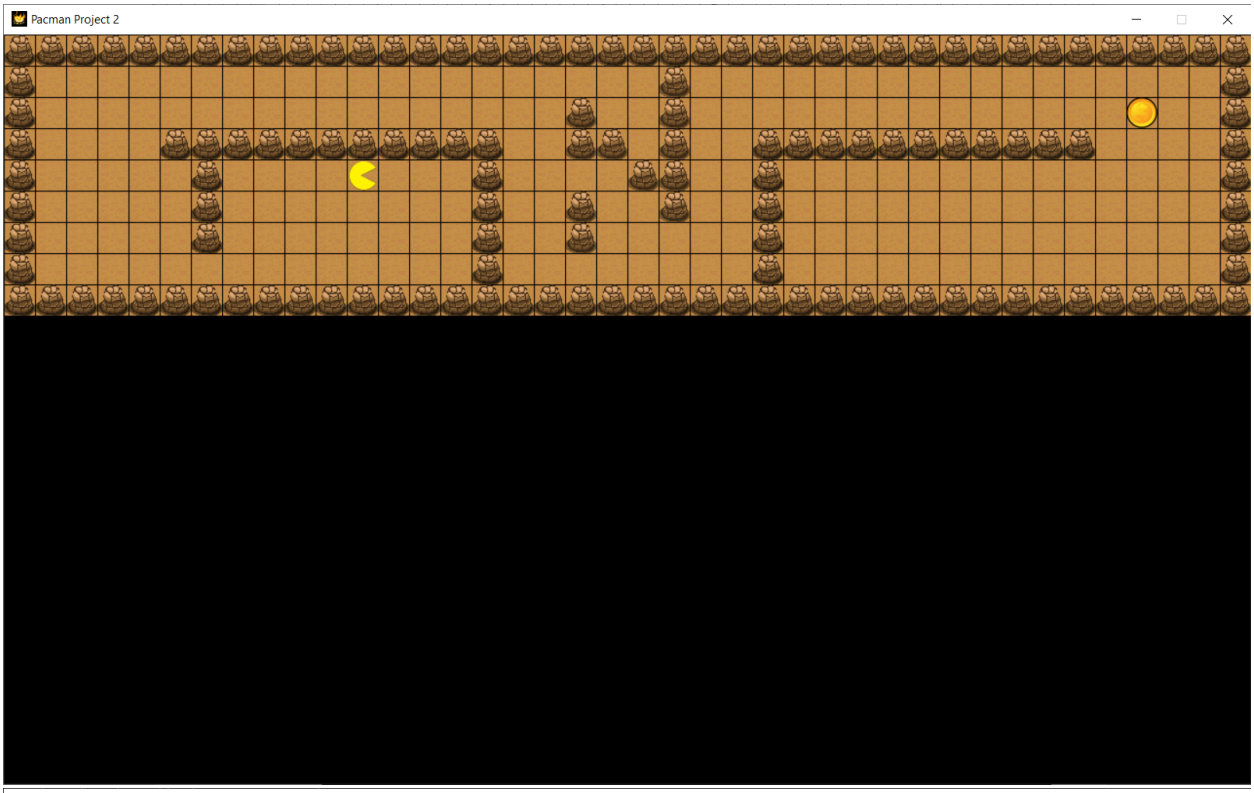
- Map3

- Map4



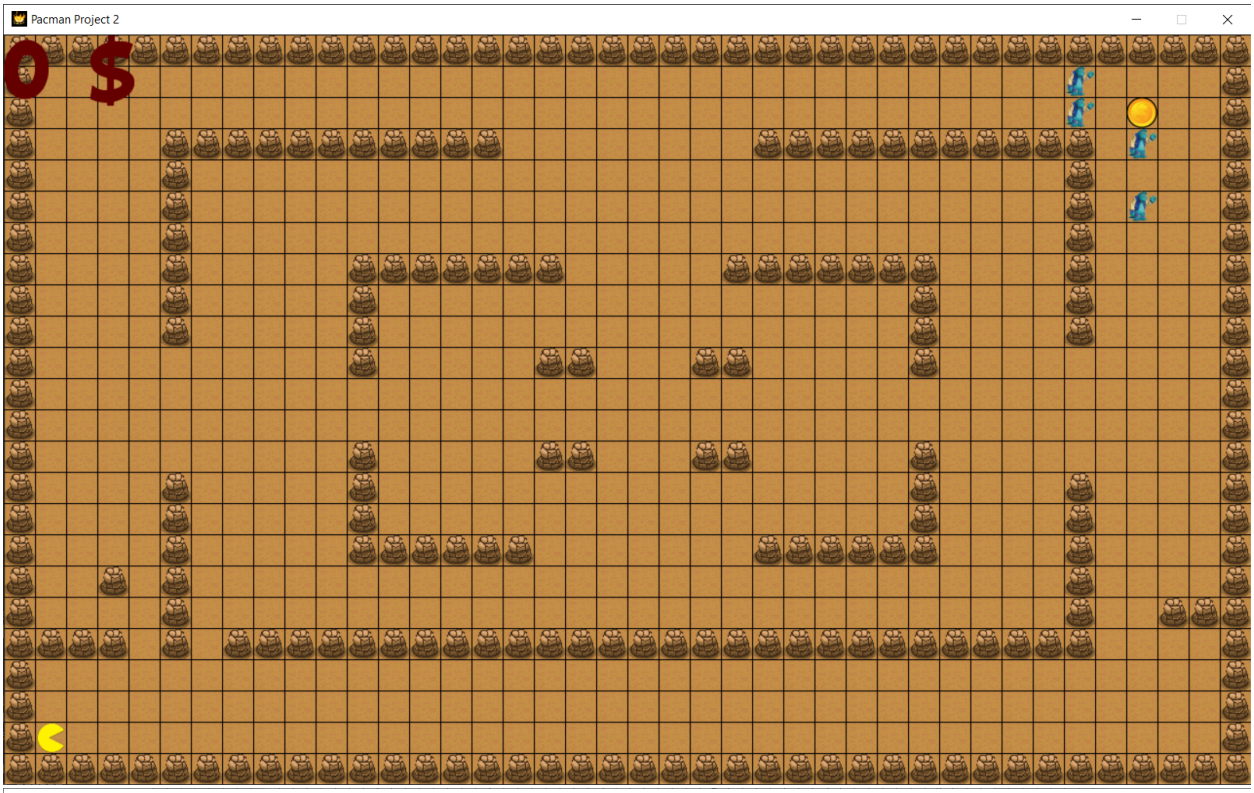


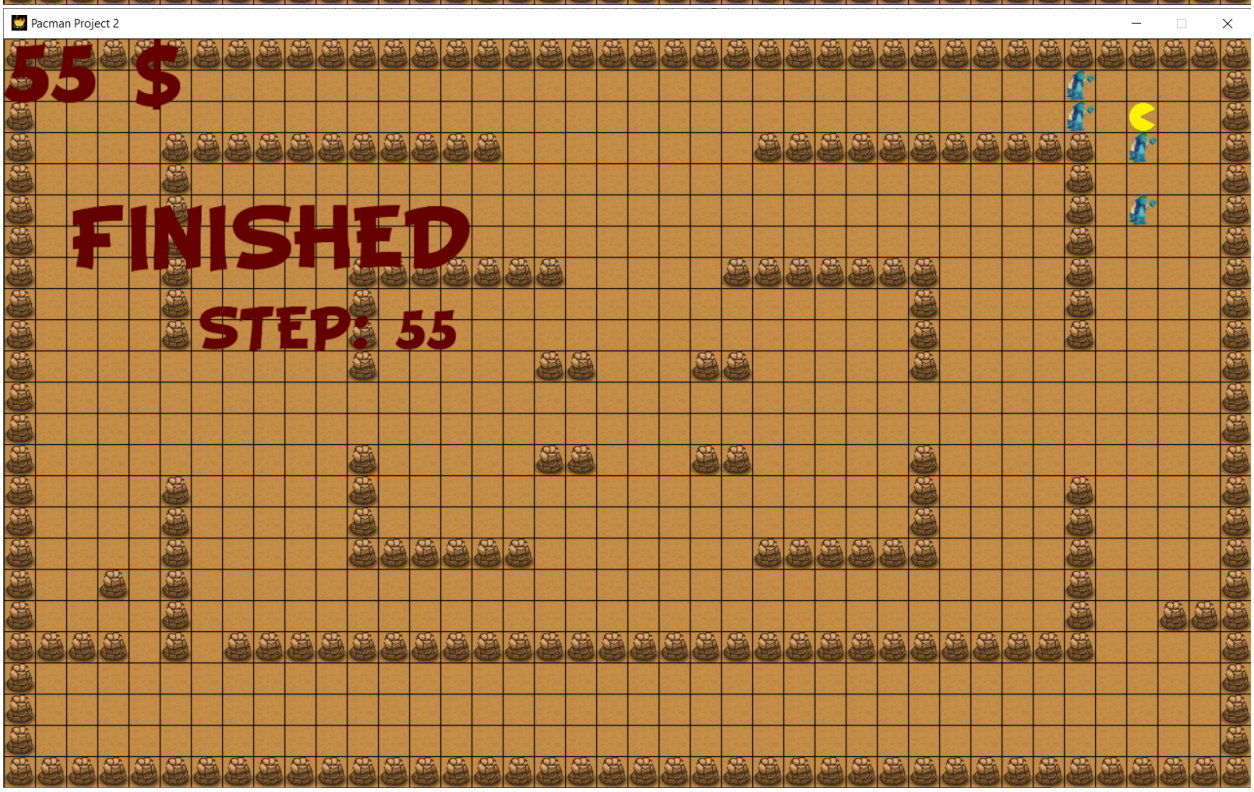
- Map 5



2. Level 2

- Map1

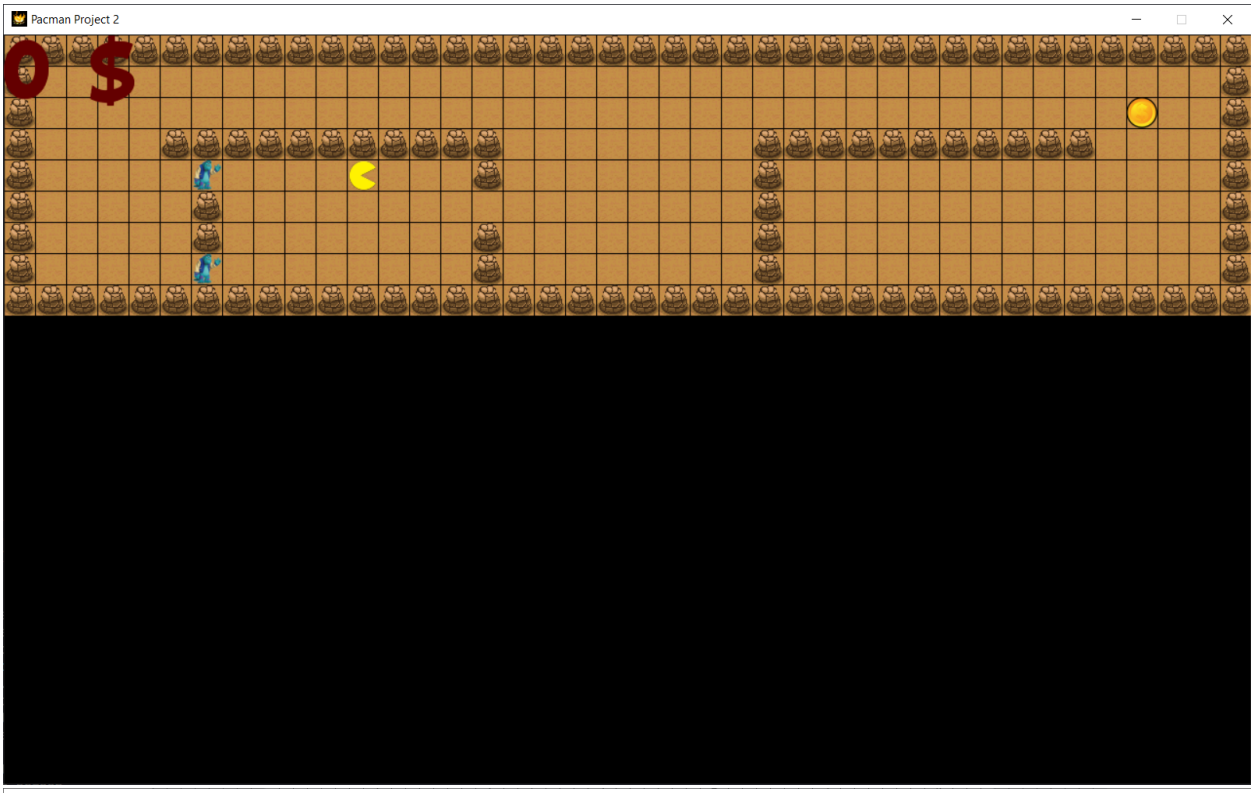


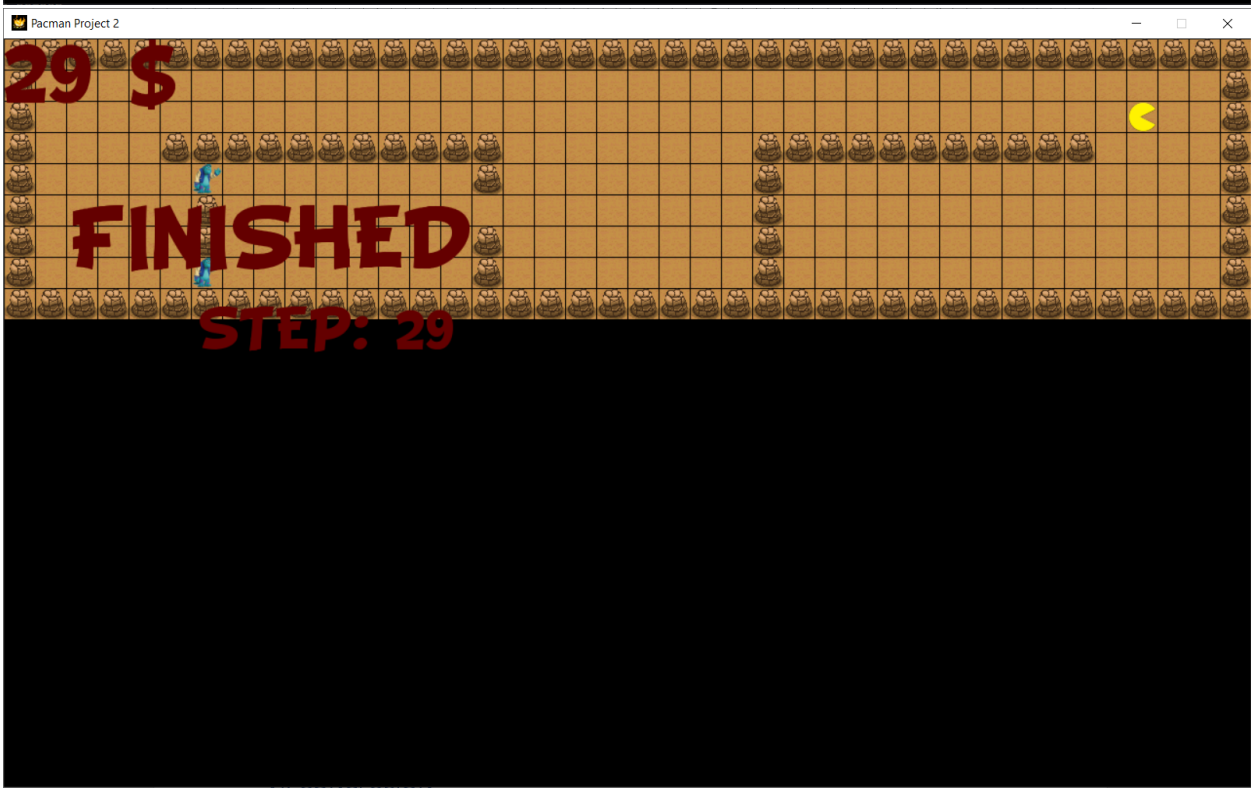


- Map2

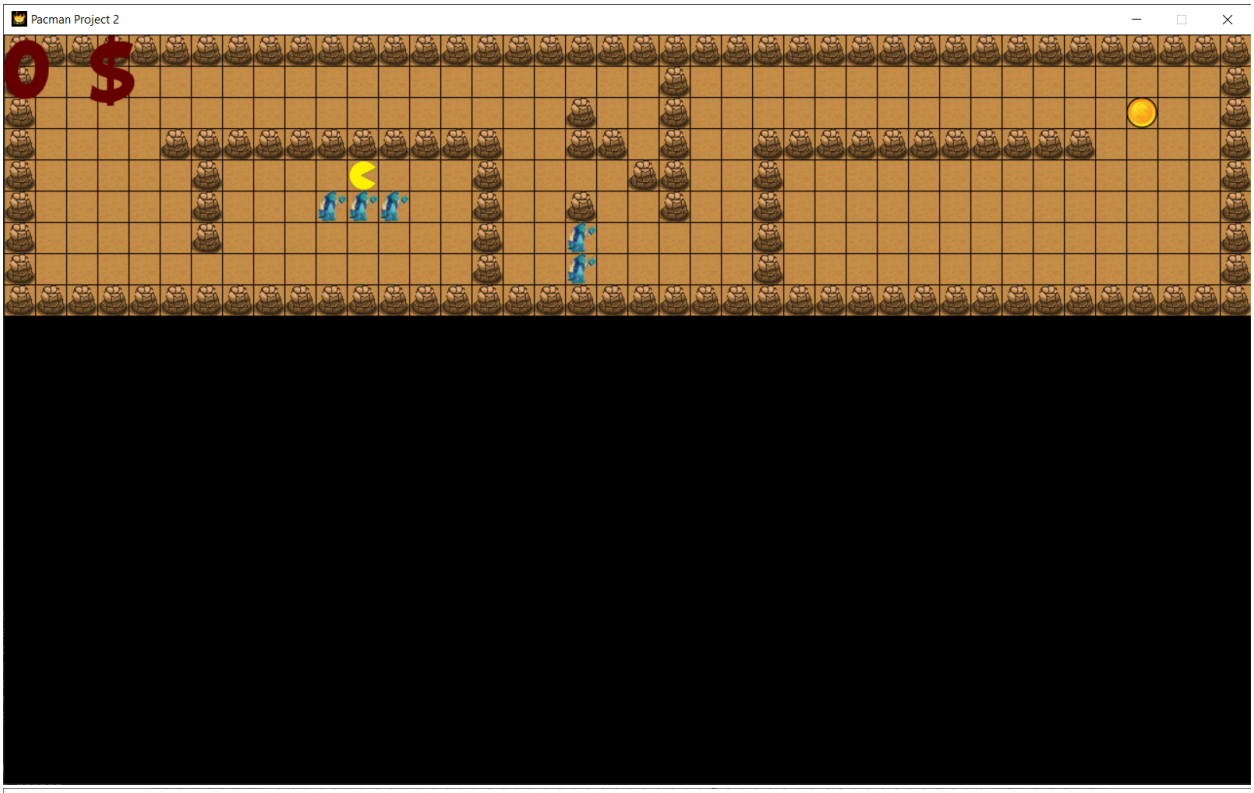
- Map3

- Map4





- Map5

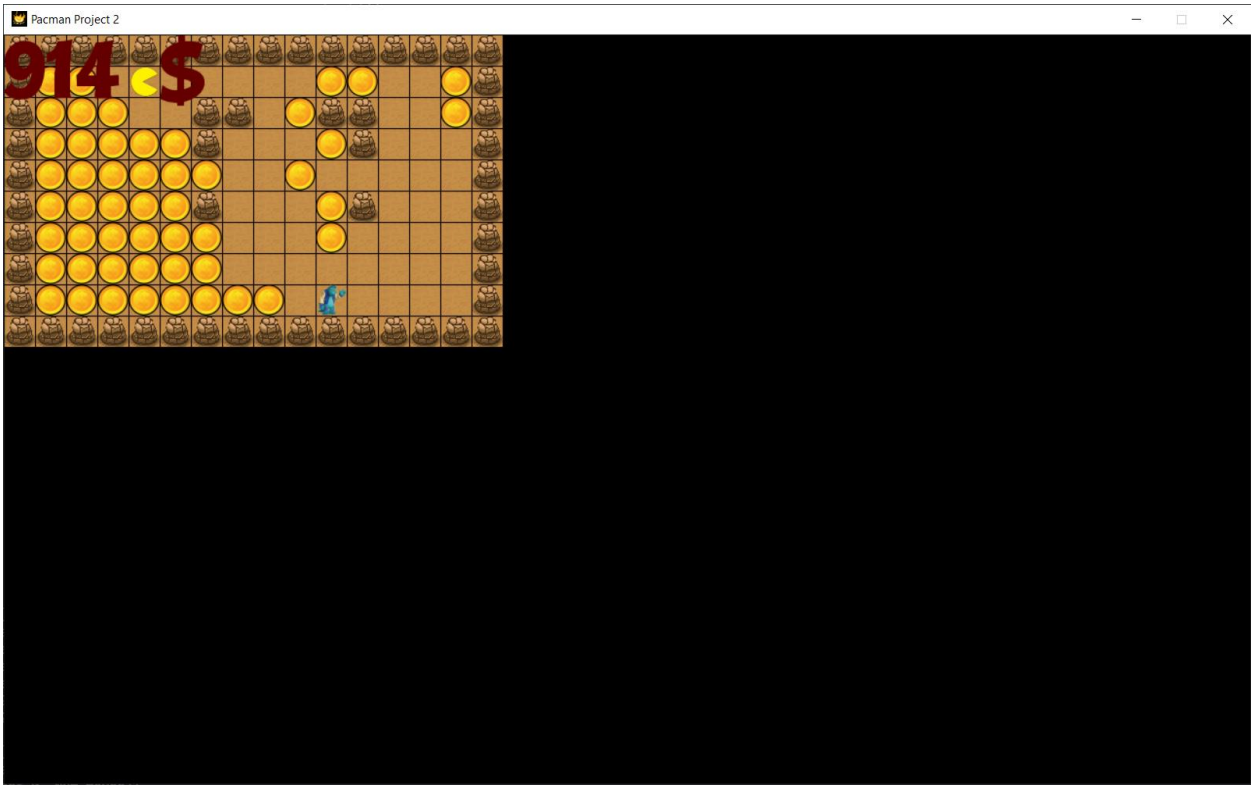


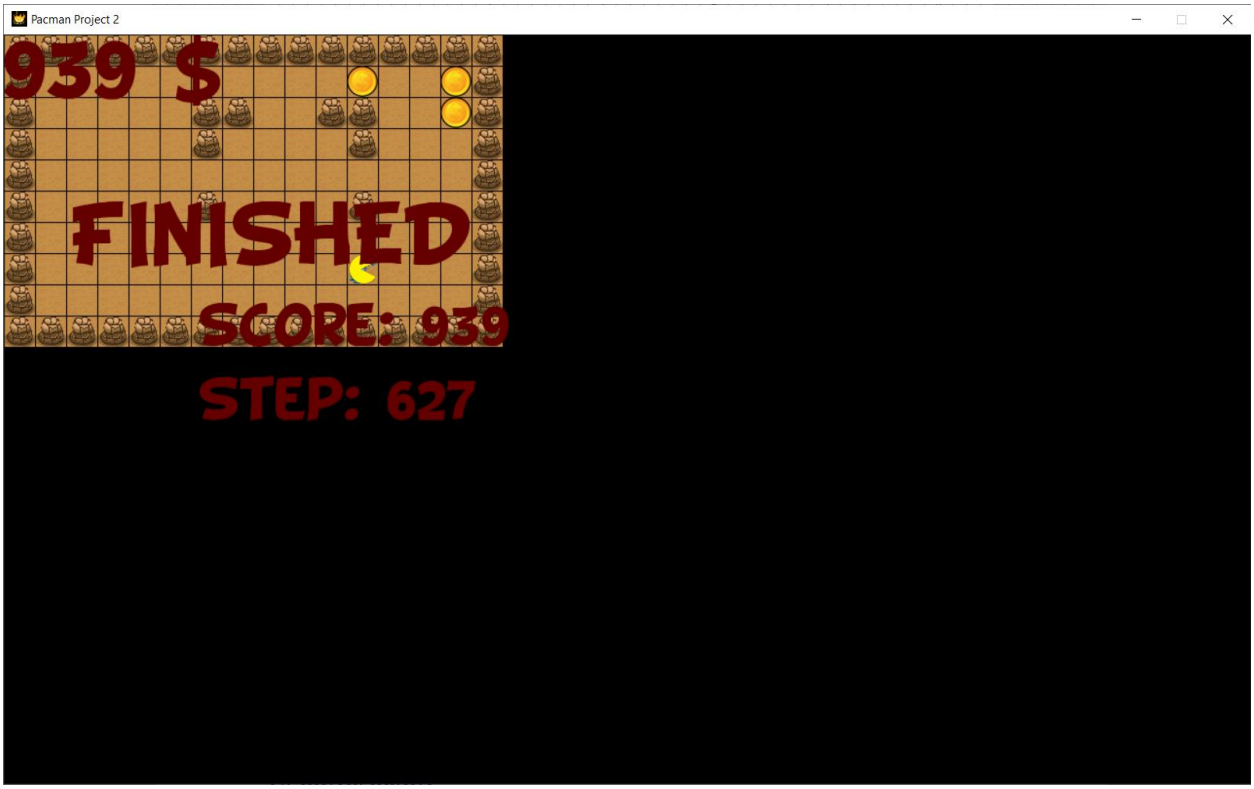
3. Level 3


Pacman Project 2
678 $


Pacman Project 2
1333 $
FINISHED
SCORE: 1333
STEP: 107

- Map2


Pacman Project 2
914 $

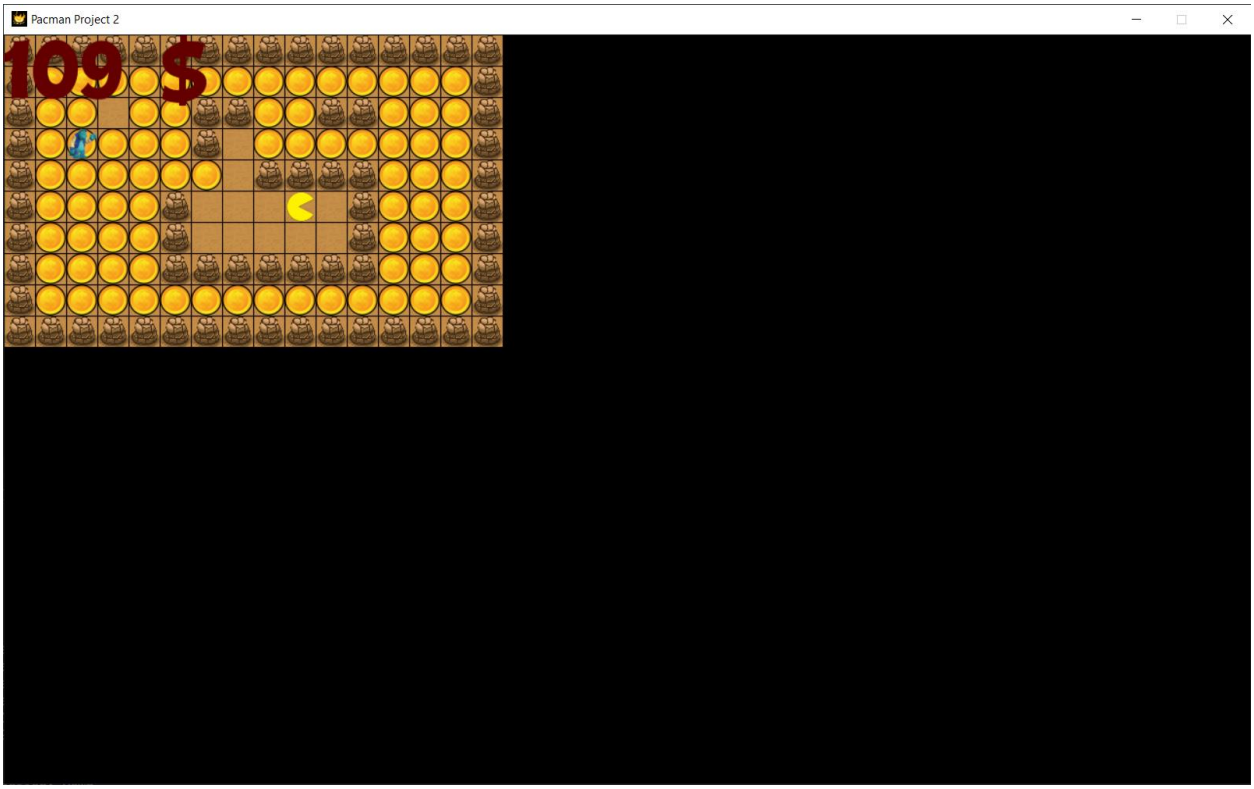

Pacman Project 2
939 $
FINISHED
SCORE: 939
STEP: 627

- Map3


Pacman Project 2
1294 $


Pacman Project 2
1383 $
FINISHED
SCORE: 1383
STEP: 163

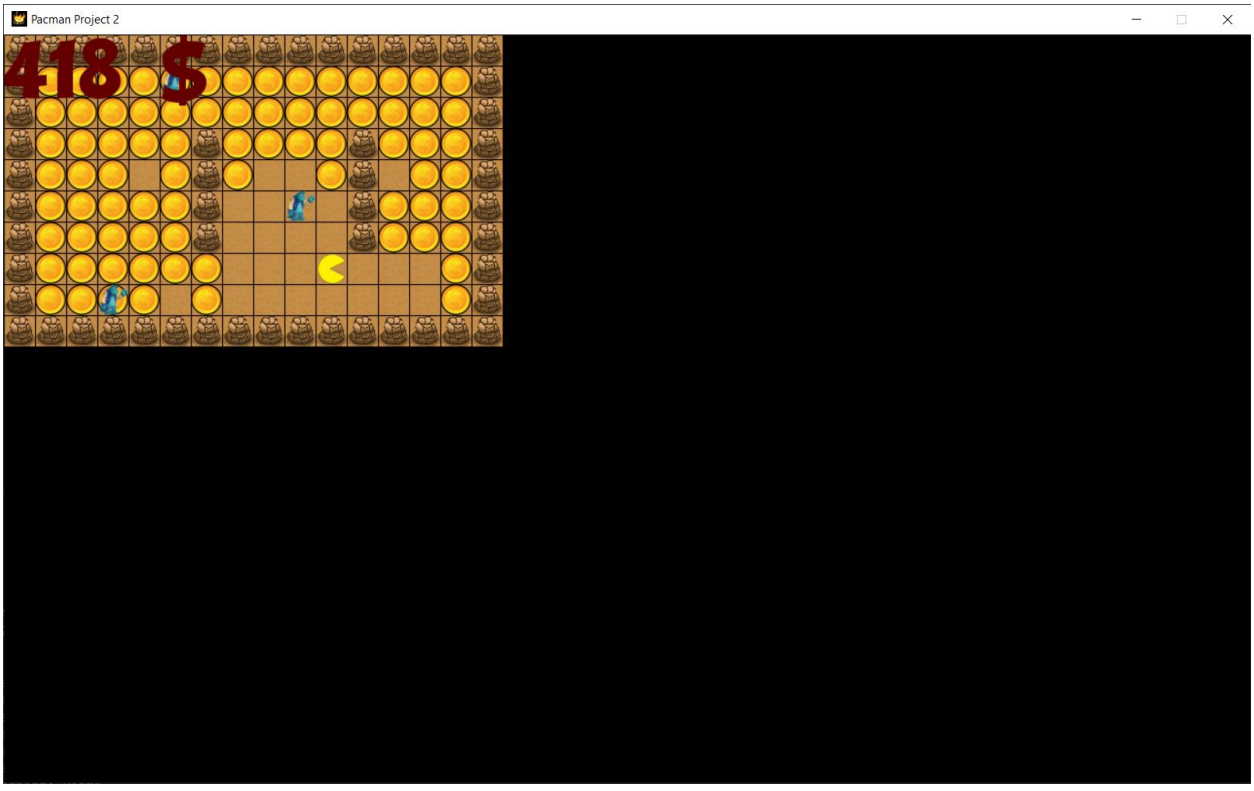
- Map4



- Map5

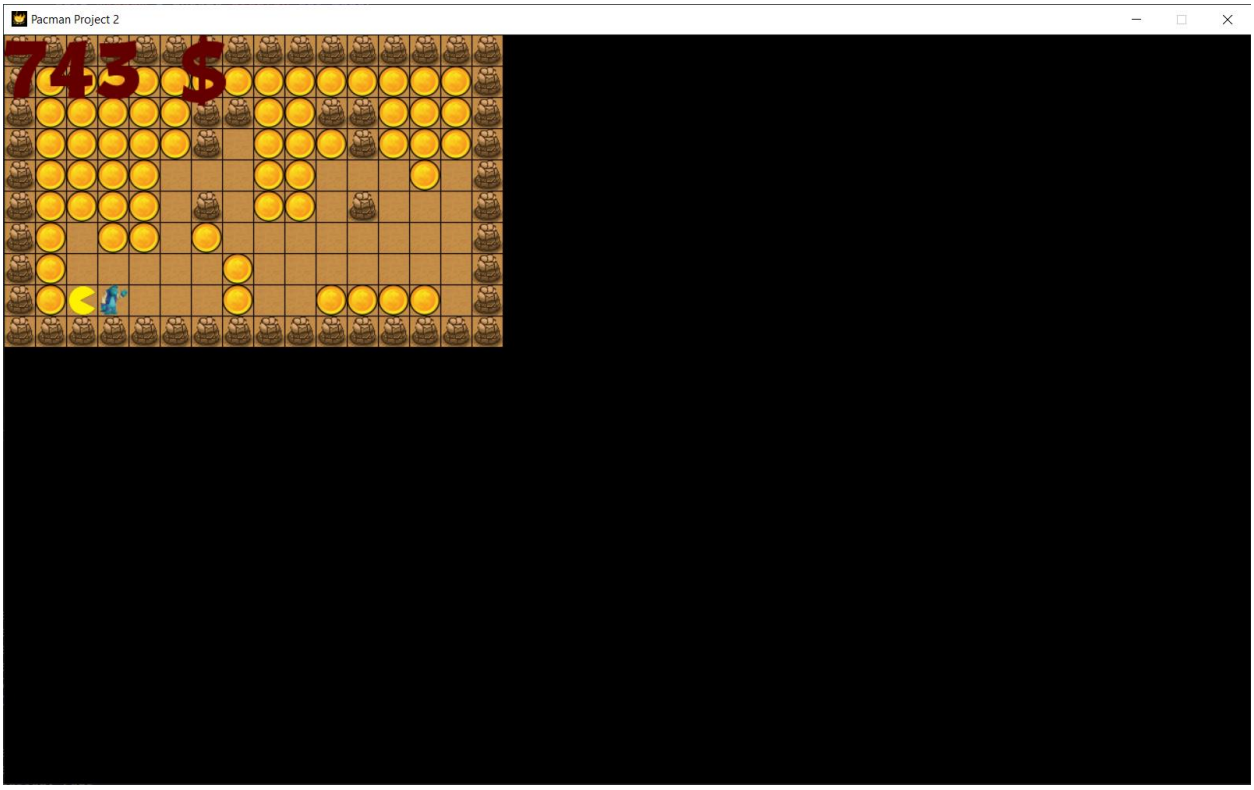
4. Level 4

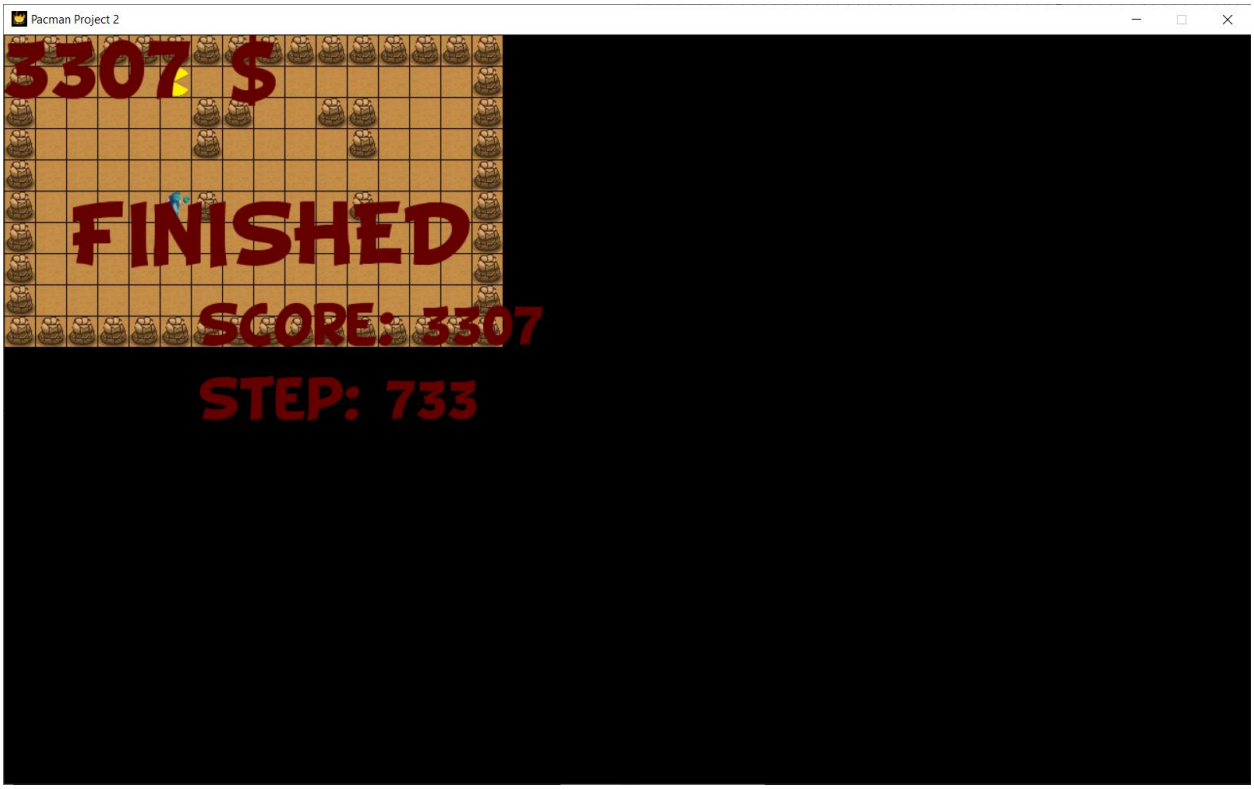
- Map1



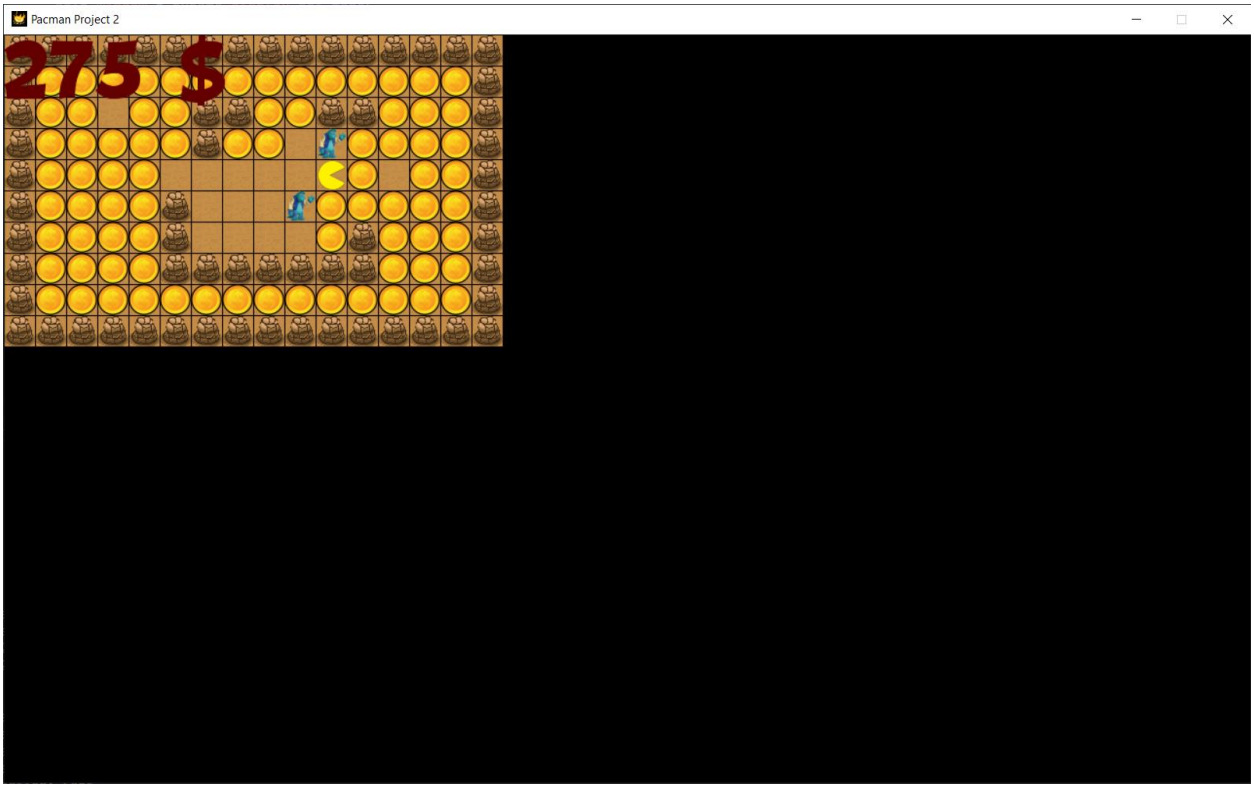

Pacman Project 2
1261 $
FINISHED
SCORE: 1261
STEP: 118

- Map 2


Pacman Project 2
743 $


Pacman Project 2
3307 $
FINISHED
SCORE: 3307
STEP: 733

- Map3


Pacman Project 2
275 $


Pacman Project 2
626 $
FINISHED
SCORE: 626
STEP: 53

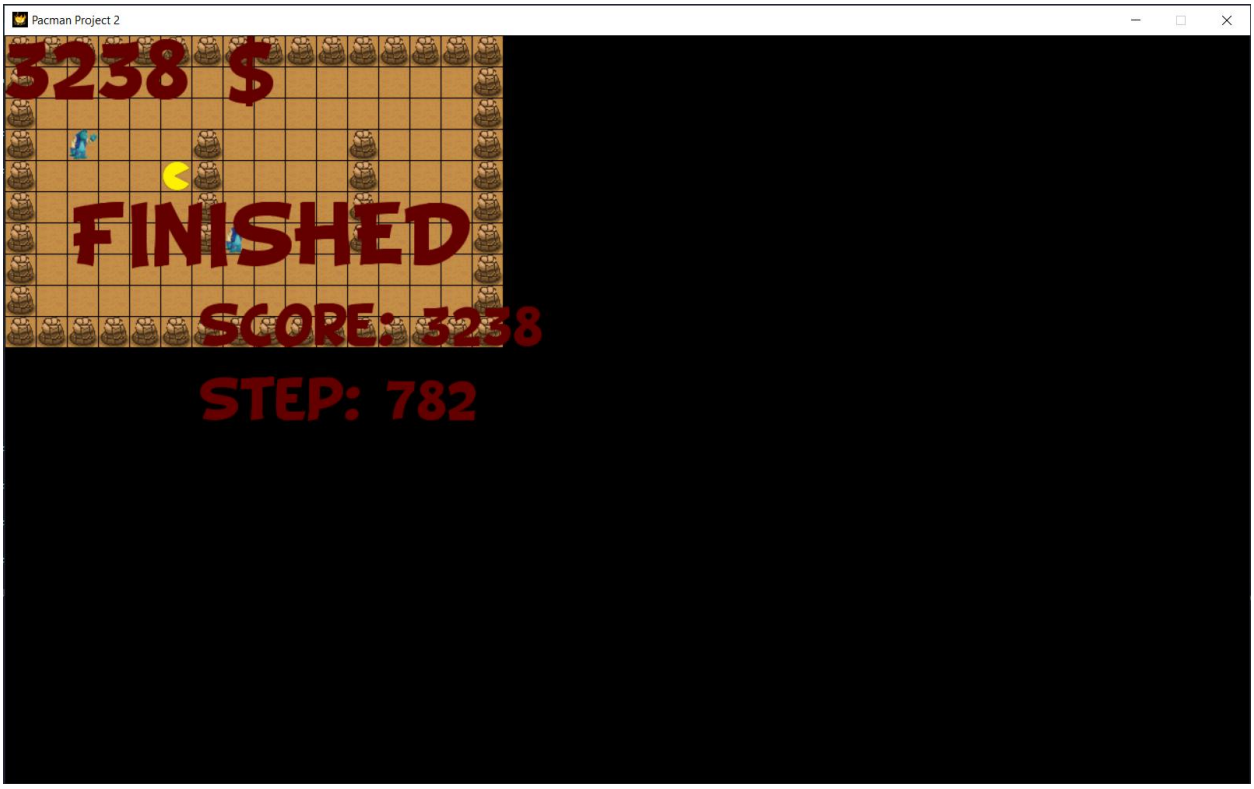
- Map4


Pacman Project 2
261 $


Pacman Project 2
850 $
FINISHED
SCORE: 850
STEP: 309

- Map 5


Pacman Project 2
145 $



VI. CONCLUSION

- Level 1, 2 use BFS pacman always eat food if have existed path
- Level 3, 4 pacman can reach a situation when have equal score (pacman can die) since it cannot eat fully point. If pacman can stand in a position in a step it can be solved
- From this project, we can conclude that the search strategy can be apply at a game to play automatically and smartly. We estimate our task done in 90% with our expectation.
- All source of this project is available on github with open source license. https://github.com/snowdence/hcmus_ai_pacman

VII. REFERENCE

[1] . AI document of CSC14003 (18CLC1)

[2] . CS188 Berkeley University AI Material http://ai.berkeley.edu/home.html

[3] . Pygame development https://www.pygame.org/wiki/tutorials