Homework 1

Tuesday, February 1st, 2022 at 11:59 pm (part 1 - 5 points)
Tuesday, February 8th, 2022 at 11:59 pm (part 2 - 95 points)
(100 points total)

Objectives

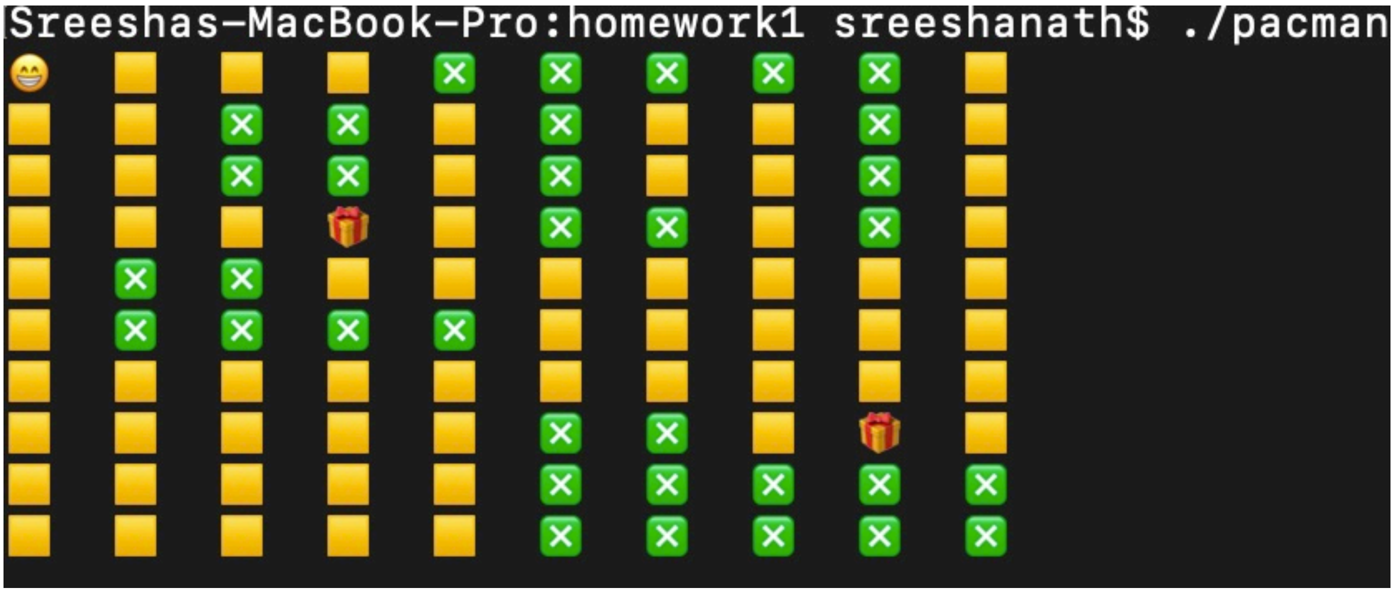
- Work with a multi-file c++ program
- Create a program that involves structs, classes, and enums
- Become (more) familiar with pointers and references
- Understand and review how objects are instantiated and how to use and manipulate them via methods and fields

Turn In:

The individual files listed here: main.cpp, Game.h, Game.cpp, Player.h, Player.cpp, Makefile. You *may* split Board into its own Board.h and Board.cpp files if you choose.

Instructions:

Your job is to implement a pacman game. We'll implement a basic, but forgiving text UI to play our game. You can think of this game as the first thing that you would implement if you were trying to implement pacman. There is a single human player who you will ask which direction they want to go each turn, and the player will walk around collecting treasures until they collect all the pellets (or die). Here is a screenshot of what the board looks like for us and a description of the different pieces:



```
Sreeshas-MacBook-Pro:homework1 sreeshanath$ ./pacman
😁 🟨 🟨 🟨 ❎ ❎ ❎ ❎ ❎ 🟨
🟨 🟨 ❎ ❎ 🟨 ❎ 🟨 🟨 ❎ 🟨
🟨 🟨 ❎ ❎ 🟨 ❎ 🟨 🟨 ❎ 🟨
🟨 🟨 🟨 🎁 🟨 ❎ ❎ 🟨 ❎ 🟨
🟨 ❎ ❎ 🟨 🟨 🟨 🟨 🟨 🟨 🟨
🟨 ❎ ❎ ❎ ❎ 🟨 🟨 🟨 🟨 🟨
🟨 🟨 🟨 🟨 🟨 🟨 🟨 🟨 🟨 🟨
🟨 🟨 🟨 🟨 🟨 ❎ ❎ 🟨 🎁 🟨
🟨 🟨 🟨 🟨 🟨 ❎ ❎ ❎ ❎ ❎
🟨 🟨 🟨 🟨 🟨 ❎ ❎ ❎ ❎ ❎
```

Here,

- the **player** is the smiley face(😁),
- **walls** are green crosses(❎),
- **pellets** are yellow squares(🟨),
- **enemies** are ghosts(👻), and
- **treasures** are the gifts(🎁).

Board initialization

Given the low probability of randomly generating a playable board for this game, you will use a static layout for the board for each game with respect to the pellets and the walls. You may create a layout of your own or use the layout shown in the image above. If you are creating a custom layout, make sure that:

- The player should be able to acquire all the pellets and the treasures
- The enemy should be able to reach the player

In other words, the walls should not cordon off the player, enemies, pellets or treasures.

Rules for placing the remaining pieces on the board:

- The player always starts in the upper left corner
- Treasures appear with a 10% chance in spaces that are not walls, or the beginning space
- You can decide where the enemies start, so long as they do not start on a wall, a

treasure, or the beginning space

Game rules

Each turn, the player should be told which directions are legal moves. Every move that does not go off the board or run into a wall is a legal move. Diagonal moves are not legal moves. The enemies(👻) appear on the board when it is initialized.

When a player moves on to a position has a pellet:

- The player will earn 1 point
- The pellet (🟨) will be replaced with a ⬜ to indicate that the pellet has already been collected. No points are to be allocated to the player if they were to move on to a position that has ⬜

If the player acquires a treasure:

- They will receive 100 points.
- Their appearance will change to indicate that they have earned a power-up. We are using 😇 to suggest the same.
- If the player moves onto a square that has an enemy, they can destroy the enemy with this power-up.
- Once the power-up is used to destroy an enemy, the appearance of the player goes back to 😁 if the player does not have any more treasures.

If the player does not have a treasure:

- If an enemy moves onto a square with the player on it, the player is destroyed.
- If the player moves onto a square with an enemy on it, the player is destroyed.

Part 1 - Diagram your Program (due Tuesday, February 1st at 11:59pm)

We've provided you with the Makefile and headers for this homework. Your first goal is to plan out how they will interact. We've given you an example for the Player object here, so your job is to complete this for Board and Game.

For each method of the given object, you should briefly describe what it does (feel free to re-word our comments into your own words) and which other methods it will need to call. The main goal here is for you to think through the structure of your program before writing it. If you find when you are implementing it, that you need to change some decisions that you made here, that is fine.

Example: Player

Method	Description
Player(const std::string name, const bool is_human)	initialize a Player with the given name and whether or not it is player
void ChangePoints(const int x)	update the points_ value according to the int passed in
void SetPosition(Position pos)	update the Player's pos_ to the new position
std::string ToRelativePosition(Position other)	translate the other position into a direction relative to the Player by comparing other with pos_

Here, Position is a user-defined type and you'll see its definition in Player.h

Note: none of the methods here rely on calling other methods—this is not the case for Board and Game!

Part 2 - Implement your Program (due Tuesday, February 8th at 11:59pm)

Step 1 — the “basic” game (80 points)

We are providing header files and a Makefile. Your job is to implement the guts of the objects and re-create this game. Your display doesn’t need to exactly match ours (use your design and text UI skills as you wish), but the functionality should match ours.

The header files that we provide currently have most of the code commented out. This is so that you can uncomment and implement as you go. You should not attempt to implement everything before compiling.

Here are two strategies that you might use when implementing this game:

1. Build your methods based on the workflow you're working on. For example: if you’re working on building a board, you could work on writing the constructor and then overriding the `<<` operator so that you can print out the Board.
2. Start with `Player`, test out the `Player` object, then move on to `Board`, then `Game`. Again, testing as you go. Most of the functionality in `Game` will depend on you having a working Board.

In either case, you should not implement more than 1 function at a time before testing it out.

The `TakeTurn` function in Game will likely be the last one that you implement, as it will depend on having all the other functions in place and working.

There are screenshots of important milestones during the run-through of the game on the last page of this write-up.

Once you have finished implementing your game so that it works with the player, move on to adding enemies to your game.

Number of enemies: Your game will have a minimum of 2 enemies. You may add more enemies to your discretion. Your enemies should be implemented in such a way that adding another does not require you to add more code other than the initial set up of the additional enemy. The movement of the enemies will be controlled by your program. After each turn where the player moves, the enemies should assume new positions when the board is loaded for the next turn. They are allowed to move on to any adjacent position as long as it is not a wall.

Note: we have used emoji to display our game, but this is not a requirement. Some systems and linux configurations do not nicely display emoji on the terminal. It is 100% acceptable to use regular ASCII characters to display your game!

Step 2 — improving the game (5 points)

The game we have described is quite basic. There is only one kind of treasure. There aren't any traps.

Once you have implemented the basic game, choose one of the following features to implement:

1. Once the enemy is destroyed by the player, spawn a new enemy at a random location on the board.
2. Add a different kind of treasure. You should make the new kind of treasure have a different appearance from the basic treasure. This treasure can be acquired only by the enemy and arms the enemy with a power upgrade. If a player runs into this treasure, this piece will behave as a wall for the player. There are 2 scenarios to consider here when the player and the enemy move on to the other's position on the board:
   i. If the enemy has a treasure and the player does not have a treasure, the enemy can still destroy the player but does not lose its treasure until it encounters the player that has a treasure.
   ii. If the enemy has a treasure and the player has a treasure, the enemy and the player are at a stalemate and move on to their next moves. Neither dies. Both, the enemy and player will lose their treasure after this move and their appearance goes back to their original self.
3. Add some danger to the board in the form of stationary traps. Make it possible for the player to lose lives and perish before completing the game. These traps could be large, they could be invisible, they could be made of fire—it is up to you.

Make sure that you have included a comment in your main.cpp describing which of these 3 options you implemented!

Step 3 — extra credit (up to 15 bonus points)

You cannot get points for extra credit unless you have completed both step 1 and step 2. We recommend choosing #3 as the feature that you implement for step 2 if you are doing #1 from the extra credit.

1. The player should have 3 lives to complete the game. At the start of each turn print how many lives the player has. The game ends when the player has 0 remaining lives. If the player loses a life, respawn the player at the same location where it died.
2. Implement a tracking logic for the enemy. Calculate the distance between the enemy and the player and move it one step closer to the enemy in each turn. You should be able to reuse this algorithm for any number of enemies.

Style and comments (10 points)

Your files and functions should have comments. We should know how to run your program and how to use your functions from your comments. Refer to the style guide on the course github to help you out here.

Your variables should have meaningful names. Your code should be easily readable. If you have complex sections of code, you should use inline comments to clarify. You should follow the style guidelines posted on our course github.

You should not have redundant code—you should use loops, conditionals, and helper functions to avoid copy+pasting code. If you find yourself copy+pasting lots of code—pause and think about your design.

General guidelines

The program that you turn in must compile. Even if your program is incomplete, make sure it compiles. (You'll get partial credit for a partially complete and compiling submission).

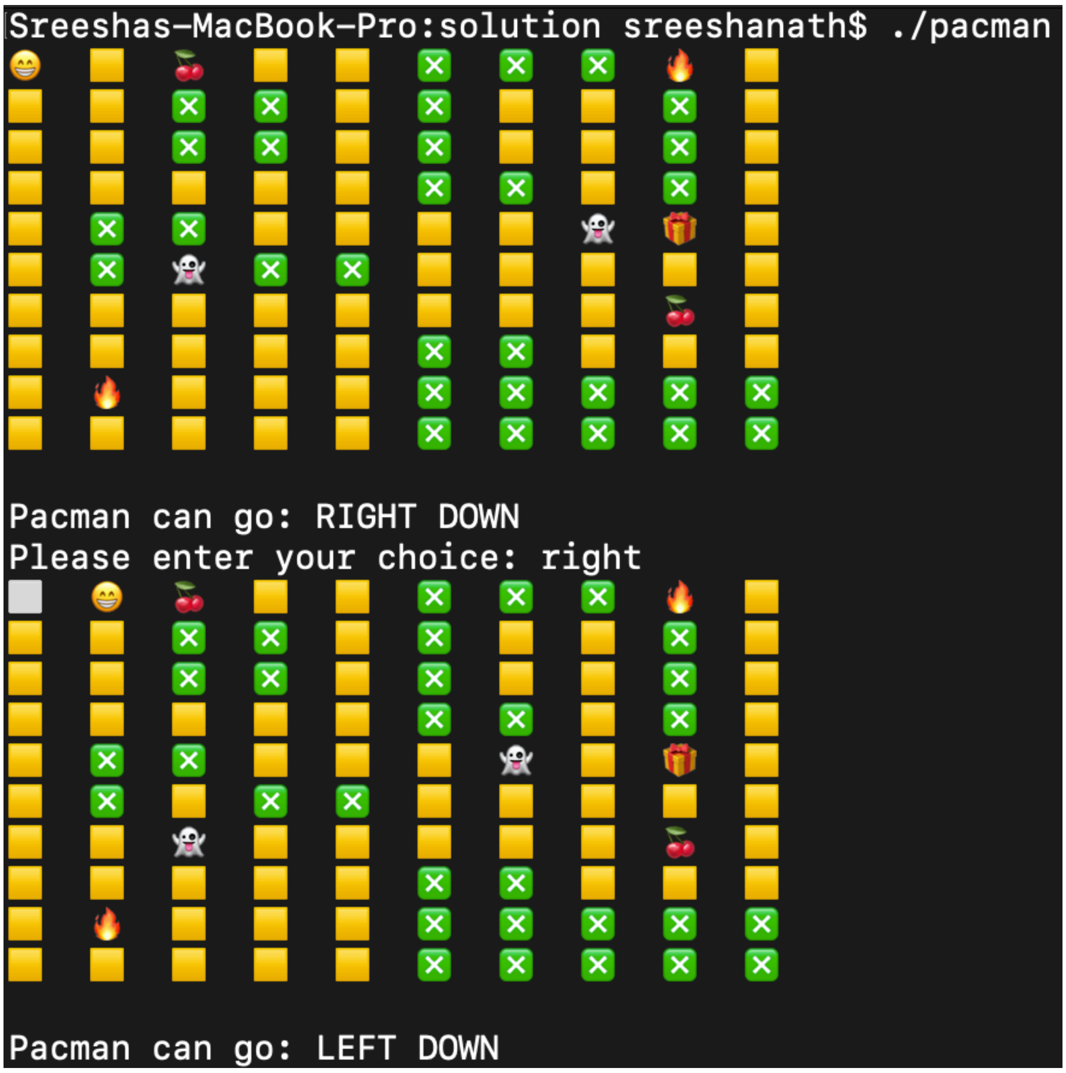
Similarly, the code that you turn in must run. You will get credit for programs that are partially complete, but you will get a hefty deduction for programs that do not run. Make sure that it runs. There will be autograder tests on Gradescope to test compilation and some functionality of your work. In addition, we will be testing your code on the coding.csel.io and the vm.

Happy coding and remember to post questions on piazza! If you'd like to see more examples of what happens in different situations, just let us know and we will provide more examples!

Some important screenshots showing the game

Image 1:

Here, you will see the updates on the pellets when the player moves. There are some additional pieces here that show you implementations for steps 2 and 3. For your reference, 🍒 is the treasure for enemies, 🔥 is a trap.



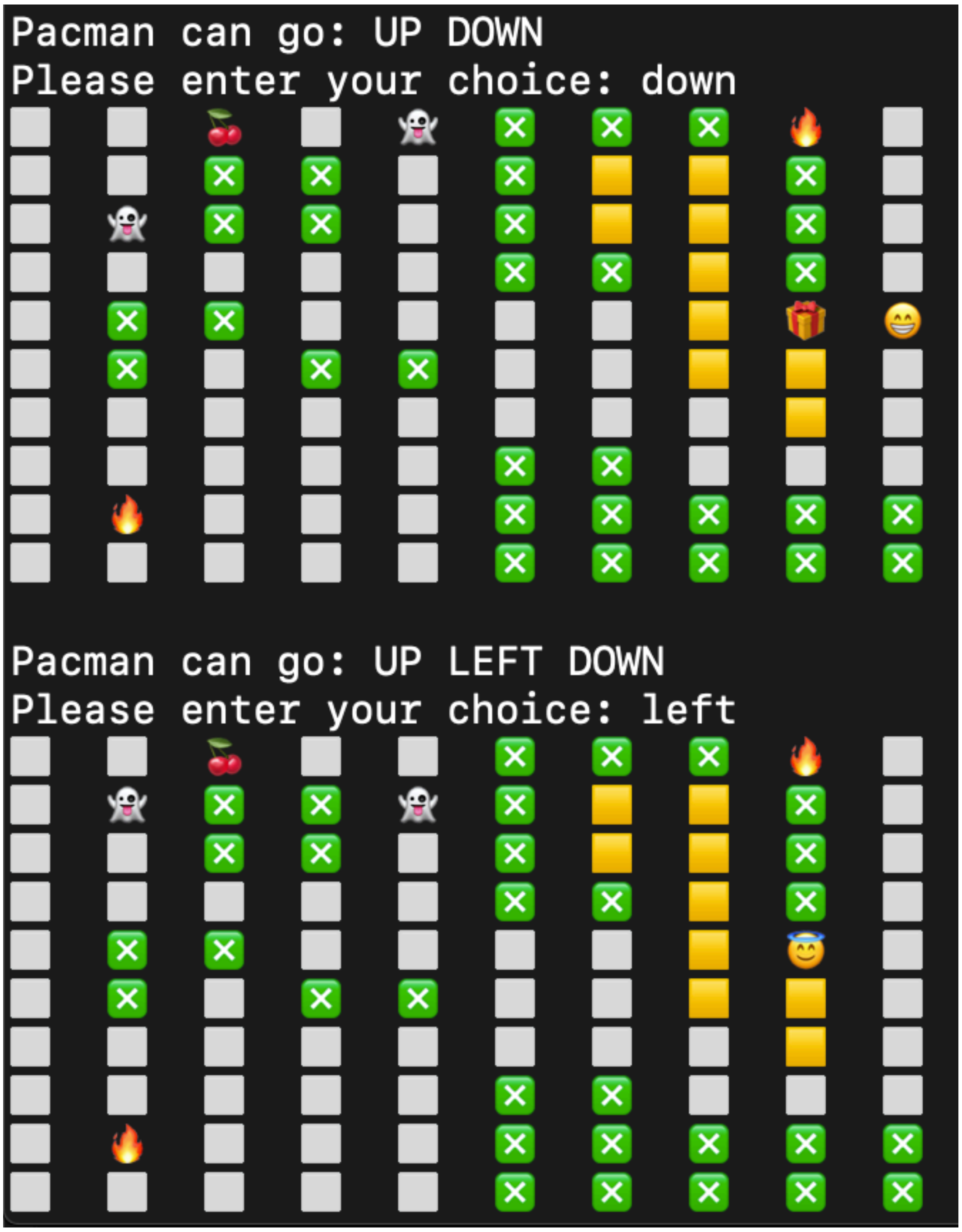
```
Sreeshas-MacBook-Pro:solution sreeshanath$ ./pacman
😁 🟨 🍒 🟨 🟨 ❎ ❎ ❎ 🔥 🟨
🟨 🟨 ❎ ❎ 🟨 ❎ 🟨 🟨 ❎ 🟨
🟨 🟨 ❎ ❎ 🟨 ❎ 🟨 🟨 ❎ 🟨
🟨 🟨 🟨 🟨 🟨 ❎ ❎ 🟨 ❎ 🟨
🟨 ❎ ❎ 🟨 🟨 🟨 🟨 👻 🎁 🟨
🟨 ❎ 👻 ❎ ❎ 🟨 🟨 🟨 🟨 🟨
🟨 🟨 🟨 🟨 🟨 🟨 🟨 🟨 🍒 🟨
🟨 🟨 🟨 🟨 🟨 ❎ ❎ 🟨 🟨 🟨
🟨 🔥 🟨 🟨 🟨 ❎ ❎ ❎ ❎ ❎
🟨 🟨 🟨 🟨 🟨 ❎ ❎ ❎ ❎ ❎

Pacman can go: RIGHT DOWN
Please enter your choice: right
⬜ 😁 🍒 🟨 🟨 ❎ ❎ ❎ 🔥 🟨
🟨 🟨 ❎ ❎ 🟨 ❎ 🟨 🟨 ❎ 🟨
🟨 🟨 ❎ ❎ 🟨 ❎ 🟨 🟨 ❎ 🟨
🟨 🟨 🟨 🟨 🟨 ❎ ❎ 🟨 ❎ 🟨
🟨 ❎ ❎ 🟨 🟨 🟨 👻 🟨 🎁 🟨
🟨 ❎ 🟨 ❎ ❎ 🟨 🟨 🟨 🟨 🟨
🟨 🟨 👻 🟨 🟨 🟨 🟨 🟨 🍒 🟨
🟨 🟨 🟨 🟨 🟨 ❎ ❎ 🟨 🟨 🟨
🟨 🔥 🟨 🟨 🟨 ❎ ❎ ❎ ❎ ❎
🟨 🟨 🟨 🟨 🟨 ❎ ❎ ❎ ❎ ❎

Pacman can go: LEFT DOWN
```

Image 2:

Here, you will see that when the player acquires the treasure, the appearance of the player is updated.

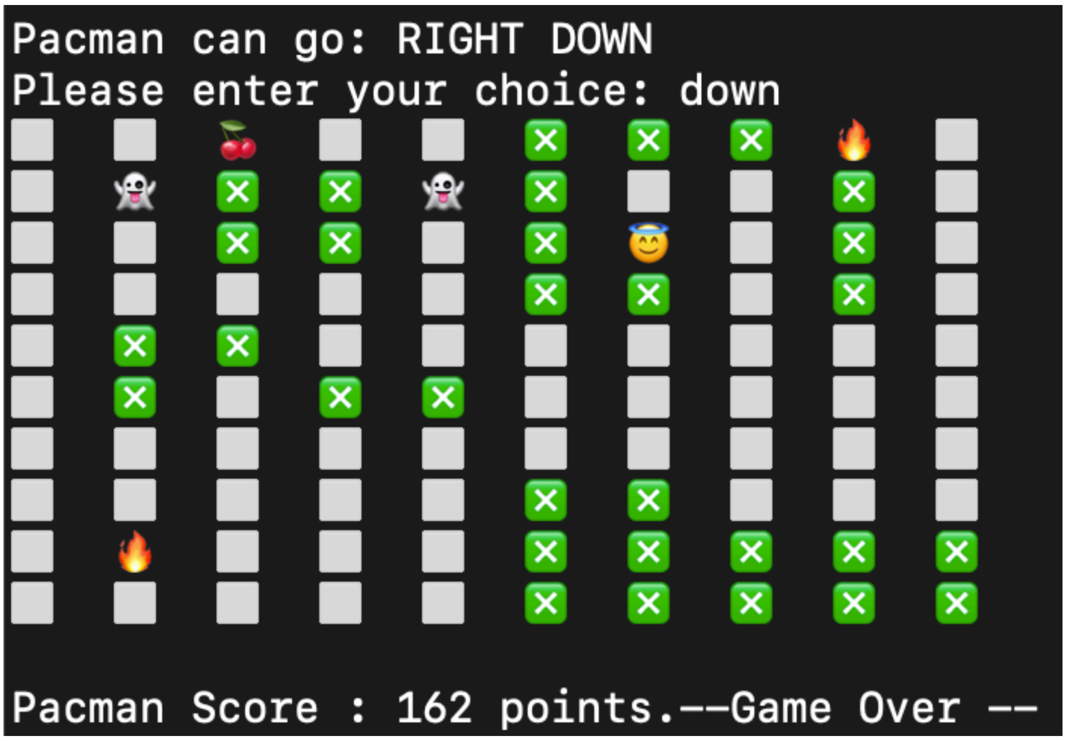


```
Pacman can go: UP DOWN
Please enter your choice: down
⬜ ⬜ 🍒 ⬜ 👻 ❎ ❎ ❎ 🔥 ⬜
⬜ ⬜ ❎ ❎ ⬜ ❎ 🟨 🟨 ❎ ⬜
⬜ 👻 ❎ ❎ ⬜ ❎ 🟨 🟨 ❎ ⬜
⬜ ⬜ ⬜ ⬜ ⬜ ❎ ❎ 🟨 ❎ ⬜
⬜ ❎ ❎ ⬜ ⬜ ⬜ ⬜ 🟨 🎁 😁
⬜ ❎ ⬜ ❎ ❎ ⬜ ⬜ 🟨 🟨 ⬜
⬜ ⬜ ⬜ ⬜ ⬜ ⬜ ⬜ ⬜ 🟨 ⬜
⬜ ⬜ ⬜ ⬜ ⬜ ❎ ❎ ⬜ ⬜ ⬜
⬜ 🔥 ⬜ ⬜ ⬜ ❎ ❎ ❎ ❎ ❎
⬜ ⬜ ⬜ ⬜ ⬜ ❎ ❎ ❎ ❎ ❎

Pacman can go: UP LEFT DOWN
Please enter your choice: left
⬜ ⬜ 🍒 ⬜ ⬜ ❎ ❎ ❎ 🔥 ⬜
⬜ 👻 ❎ ❎ 👻 ❎ 🟨 🟨 ❎ ⬜
⬜ ⬜ ❎ ❎ ⬜ ❎ 🟨 🟨 ❎ ⬜
⬜ ⬜ ⬜ ⬜ ⬜ ❎ ❎ 🟨 ❎ ⬜
⬜ ❎ ❎ ⬜ ⬜ ⬜ ⬜ 🟨 😇 ⬜
⬜ ❎ ⬜ ❎ ❎ ⬜ ⬜ 🟨 🟨 ⬜
⬜ ⬜ ⬜ ⬜ ⬜ ⬜ ⬜ ⬜ 🟨 ⬜
⬜ ⬜ ⬜ ⬜ ⬜ ❎ ❎ ⬜ ⬜ ⬜
⬜ 🔥 ⬜ ⬜ ⬜ ❎ ❎ ❎ ❎ ❎
⬜ ⬜ ⬜ ⬜ ⬜ ❎ ❎ ❎ ❎ ❎
```

Image 3:

Once the player has collected all the pellets and treasures, the game concludes and the player's final score is displayed.



```
Pacman can go: RIGHT DOWN
Please enter your choice: down
⬜   ⬜   🍒   ⬜   ⬜   ❎   ❎   ❎   🔥   ⬜
⬜   👻   ❎   ❎   👻   ❎   ⬜   ⬜   ❎   ⬜
⬜   ⬜   ❎   ❎   ⬜   ❎   😇   ⬜   ❎   ⬜
⬜   ⬜   ⬜   ⬜   ⬜   ❎   ❎   ⬜   ❎   ⬜
⬜   ❎   ❎   ⬜   ⬜   ⬜   ⬜   ⬜   ⬜   ⬜
⬜   ❎   ⬜   ❎   ❎   ⬜   ⬜   ⬜   ⬜   ⬜
⬜   ⬜   ⬜   ⬜   ⬜   ⬜   ⬜   ⬜   ⬜   ⬜
⬜   ⬜   ⬜   ⬜   ⬜   ❎   ❎   ⬜   ⬜   ⬜
⬜   🔥   ⬜   ⬜   ⬜   ❎   ❎   ❎   ❎   ❎
⬜   ⬜   ⬜   ⬜   ⬜   ❎   ❎   ❎   ❎   ❎

Pacman Score : 162 points.--Game Over --
```