Pac-Man

Final Design Document

COMP 504

Rice University

Team Team

Jiaxin Wu, Lee-Hsun Hsieh, Shuai He,

Soham Pajwani, Xiangnan Chen, Yi Zhang

Contents

Section 1 - Project Description

Pac-Man Heroku link: https://pacman-final-team-team.herokuapp.com/

Section 2 - Overview

2.1 Purpose

The purpose of this project is to design and implement a user-friendly online Pac-Man application that satisfies all customer requirements listed below by utilizing OOD strategies learned from the COMP 504 course.

2.2 Requirements

Requirements are cited from the rubric of assignment 8.

1. Ghosts will use some behavior to move toward/away from Pac-Man. They won't all use the same behavior.
2. Periodically, a piece of fruit will appear that will be worth 100 points.
3. Pac-Man does not move when colliding with a wall
4. Pac-Man should be able to exit one side of the game board and enter in on the other side
5. When Pac-Man eats the piece of fruit, the fruit disappears
6. When Pac-Man eats small dots, the dots disappear. Each small dot is 10 points.
7. When Pac-Man eats large dots, the ghosts turn dark blue and then start flashing (blue and white colors) for a small period of time. Each large dot is 50 points.
8. If Pac-Man collides with a dark blue or flashing ghost, the ghosts become two eyes and travel quickly to the square box in the middle of the screen. For a single large dot, the first ghost Pac-Man collides with is worth 200, the second is worth 400, the third is worth 800, and the fourth is worth 1600.
9. if Pac-Man collides with a non-dark blue or non-flashing ghost, Pac-Man loses 1 life
10. Pac-man starts with 3 lives.
11. The game ends if Pac-Man loses all 3 lives. There should be a "Game Over" message shown on the game.
12. Pac-Man advances to the next level if Pac-Man eats all the dots before losing 3 lives. Each level should become more difficult.
13. Keep track of Pac-man's score for the game. The score doesn't need to be saved when starting a new game.
14. The Pac-Man game should be extensible in some way that can be selected by a user.

Section 3 - Use Cases

Use cases diagram below gives a brief review of how users will interact with the application. Detailed use case analysis of *Pac-Man* can be categorized into the following categories.

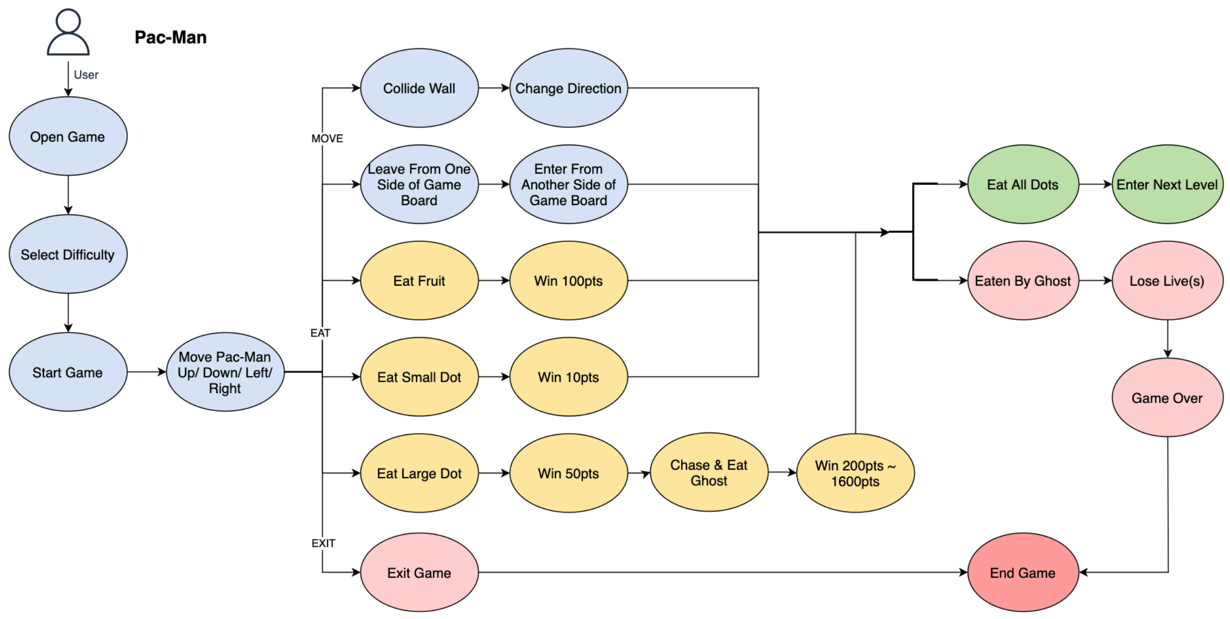

Figure 1. Use Case Diagram.

3.1 User (Pac-Man)

- Play game
- Select difficulty level (i.e. number of ghosts, the velocity of ghosts)
- Move Pac-Man left, right, up, down
- Collide wall
- Exit one side of the game board and enter from another side of the game board.
- Eat fruit
- Eat small dots
- Eat large dots
    - Chase ghost
    - Eat ghost
- Run away from ghost
- Eat all dots and enter next level
- Eaten by ghost
    - Lose life
- Game over
- Check final score
- Exit game

3.2 Ghosts

- Move in four directions
- Collide wall
- Leave home
- Chase Pac-Man
- Run away from Pac-Man
- Move randomly
- Eaten by Pac-Man
    - Become eyes
    - Run back to home
- Turn dark blue and start flashing

Section 4 - GUI Design

Following diagrams are demonstrations of the game interface of the Pac-Man game. Users will be directed to the main page (Figure 2) once they open the website. Top left corner is showing some useful information including score and current left lifes. There also are some ways for users to customize their game difficulty. For example, they can select the number of ghosts, wall styles, and difficulty levels. After setting up the game difficulty, users can click "Play" to play the game. There will be 240 small dots and 4 large dots in the map, and the user will automatically enter the next level after eating all dots. There will also be fruit popping up in the map periodically. After users finish the game, they can check the highest score in top left corner.

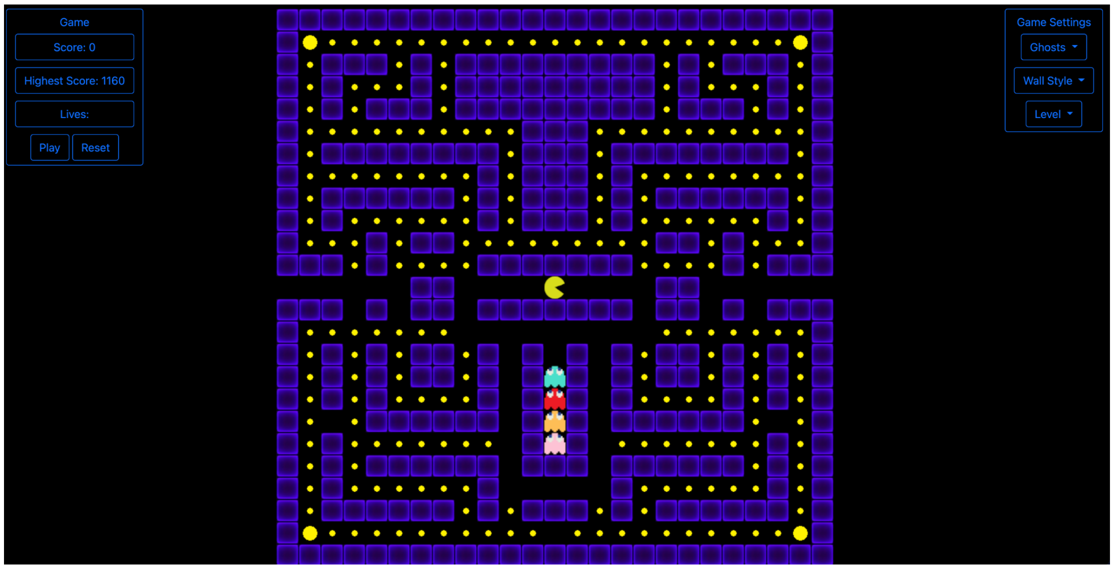

Figure 2. Main Page -- Level 1.

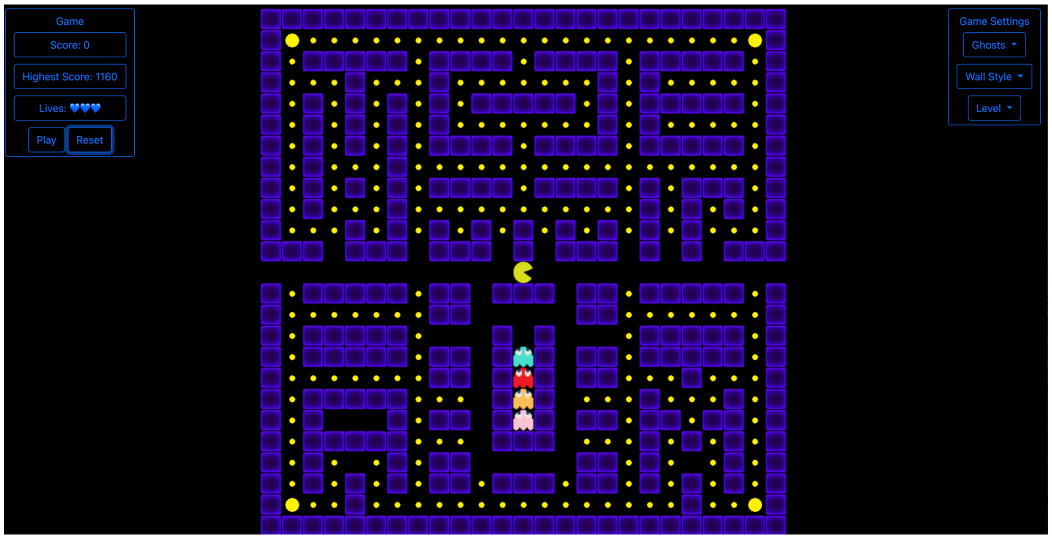

Figure 3. Main Page -- Level 2.

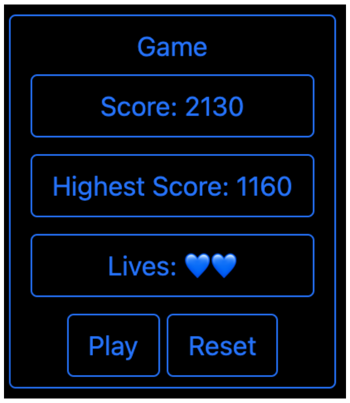

Figure 4. Score and Lives.

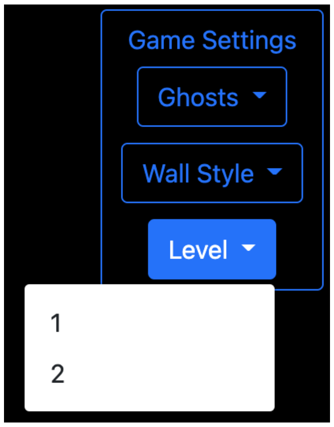

Figure 5. Select Difficulty Level.

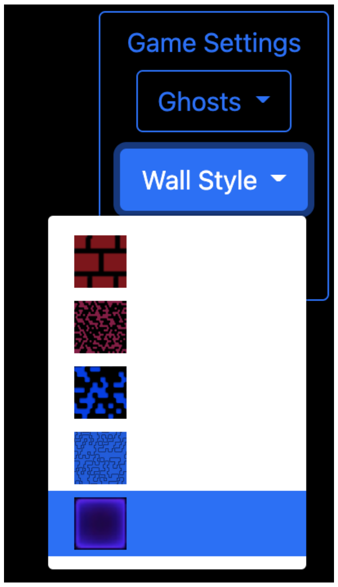

Figure 6. Select Wall Style.

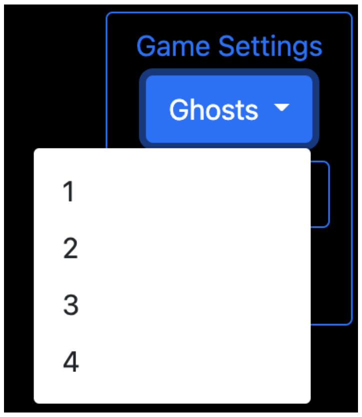

Figure 7. Select Ghost Number.

Figure 8. Game Over Screen.

Section 5 - Software Domain Design

5.1 UML Diagram

Attached is a UML diagram that shows the connections between all interfaces and classes. Detailed explanations are in the following sections.

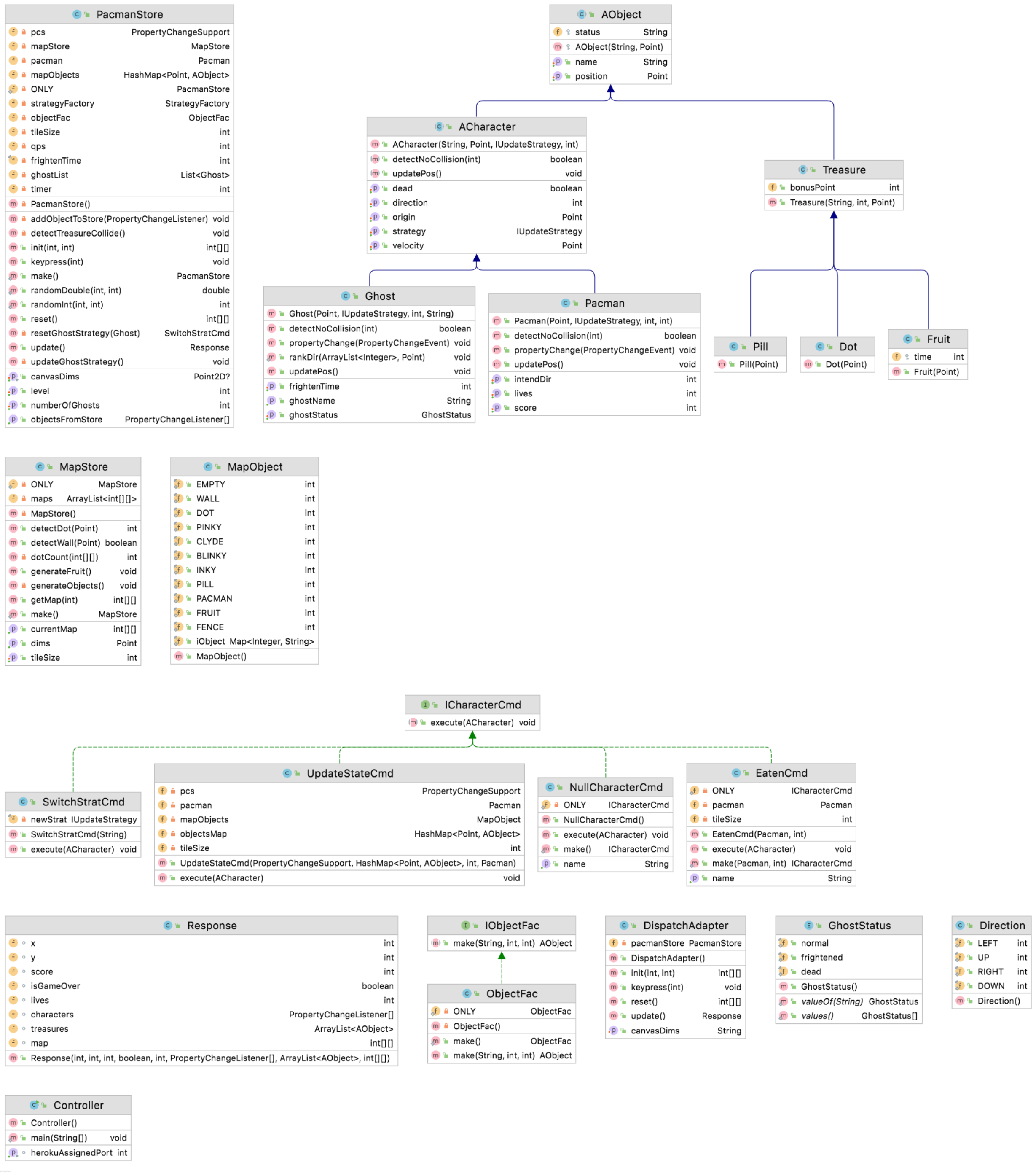

Figure 9. UML Diagram.

5.2 Design Patterns

Team Team decided to take advantage of some design patterns learned from the course to make the application more robust and scalable. Six main design patterns are used: Union Design Pattern, Factory Design Pattern, Singleton Design Pattern, Command Design Pattern, Observer Design Pattern, and Strategy Design Pattern.

- **Union Design Pattern** is widely used in the project especially when lots of concrete classes are sharing common fields and methods. For example, *PacMan* and *Ghost* are all inherited from the abstract class *Character* since they share many parameters and methods. Also, *Dot, Pill,* and *Fruit* are all implementing the abstract class *Treasure* since they share many common fields.
- **Factory Design Pattern** is used for creating *Character*, *Treasure, Command,* and *Strategy* objects because it adds an extra layer of abstraction to hide the logic and implementation of creating these objects from the client.
- **Singleton Design Pattern** is used to prevent other objects from instantiating their own copies of the *Factory* and *Store* object.
- **Command Design Pattern** is used for updating all game information (current locations, status of various objects, strategies for Pac-man, and ghosts).
- **Observer Design Pattern** is used where *Character* objects were treated as observers. This allows sending data to other objects effectively without any change in itself and makes frequent updating *Character* and *Treasure* objects more efficient.
- **Strategy Design Pattern** is used to update different characters using different strategies. For example, ghosts will have *ChaseStrategy, ScatterStrategy, RandomStrategy, etc.*

5.3 Software Application Domain

This section describes the design and functionalities of each back-end module of the application.

5.3.1 Controller

PacManController will be in charge of communicating with the dispatch adapter based on requests received from view. It includes but is not limited to the following endpoints.

Metho d	**End Point**	**Parameters**	**Returns**	**Description**
POST	/init	{ “level”: uint, “numberOfGhost”: uint, “ghostSpeed”: uint}	{ “map”: arr[m][n] }	Initialize the game and generate a game board based on users’ choice.
POST	/update		{ “map”: (i,j), “score”: uint, “isGameOver”: boolean, “lives”: uint, characters:[Pacman, Ghost1, Ghost2,...], treasures:[fruit,], }	Update the game board.
GET	/reset			Reset the game board.

GET	/highestScore		{“highest score”: highest score}	Get the highest score of the user.
POST	/keypress	{ “keypress”: num } where num in [37,38,39,40]	None	Send keypresses to the backend.

5.3.2 Dispatch Adapter

PacManAdapter is responsible for communicating between the frontend and the

Method	**Description**
setCanvasDims(String dims)	Set canvas dimensions.
update()	Update game world.
updateLevelInfo (String strat, String interact, String type)	Update level of the game.
reset(String strat)	Reset the strategy.
init()	Initialize the world.

5.3.3 Interfaces

Pac-man has three interfaces: *ICmdFac, IObjCmd, and IStrategy.*

5.3.3.1 ICharacterCmd

IObjCmd is an interface for passing commands to a *Character* object.

Method	**Description**
execute(Character context)	Pass command to a *Character* object.

5.3.3.2 IC2CCmd

IC2CCmd is an interface for passing *Character* to *Character* interactions commands.

Method	**Description**
execute(Character context1, Character context2)	Pass commands to two *Character* objects if they have interacted.

5.3.3.3 IC2TCmd

IC2TCmd is an interface used for passing *Character* to *Treasure* interaction commands.

Method	**Description**

execute(Character context1, Treasure context2)	Pass commands to a *Character* object and a *Treasure* object if they have interacted.

5.3.3.4 IStrategyCmd
IStrategy is an interface used for different kinds of strategy.

Method	**Description**
make(String type)	Make a corresponding strategy.

5.3.3.5 IUpdateStrategyCmd
IStrategy is an interface used for different kinds of strategy.

Method	**Description**
getName()	Get the name of the strategy.
updateState(Character context)	Update state of a *Character* object.
updateState(Character context, Character pacman)	Update state of a *Character* object and *PacMan*.

5.3.4 Abstract Classes
There will be two abstract classes -- *Character* and *Treasure*. They are determined to be abstract classes because there will be different types of characters (Pac-man and ghosts) and different types of treasures (small dot, large dot, fruit, and pill).

5.3.4.1 Character
Character is an abstract class for characters (Pac-man and ghosts) that contains the following fields and methods.

Field	**Type**	**Description**
strategy	IUpdateStrategy	Strategy of the character.
position	Point2D	Current Position of the character.
status	String	Eaten, normal, dead
direction	int	An integer representation of the current facing direction of Pac-man. 0--up, 1--left, 2--down, 3--right
velocity	Point2D.Float	Current velocity of the Pac-man.

Method	**Description**
detectCollision(Character char)	Detects collision between walls and the *Character* object.

5.3.4.2 Treasure

Treasure is an abstract class for “treasures” in the game including small dots, large dots, fruit, and pill.

Field	**Type**	**Description**
bonusPoint	int	Bonus score of the treasure (small dots, large dots, and fruit).
effectTime	int	Effect time for the treasure. For example, how long will ghost keep flashing after eating large dots, and how often will fruit appear.
position	Point2D	Position of the treasure.

5.3.5 Concrete Classes

Following are some concrete classes the program has. The design doc only briefly goes through concrete classes.

5.3.5.1 Pacman & Ghost & CharacterFac

Pacman and *Ghost* are concrete classes that implement the *Character* class. *CharacterFac* is the factory class for making different *Character* objects.

5.3.5.2 Wall & Fruit & Pill & Dot & TreasureFac

Wall, Fruit, Pill, and Dot are concrete classes that implement the *Treasure* class. *TreasureFac* is the factory class for making different *Treasure* objects.

5.3.5.3 Map & MapFac

Map class stores all pre-designed game boards in several 2D array representations, where each cell represents the location of all objects shown on the canvas. *MapFac* is the factory class for making different *Map* objects.

5.3.5.4 CollideCmd & C2CCmdFac & NullC2CCmd

CollideCmd takes charge of when two *Character* objects collide. *C2CCmdFac* is the factory for making *Character* to *Character* commands. *NullC2CCmd* ensures the robustness of the game.

5.3.5.5 EatenCmd & C2TCmdFac & NullC2TCmd

EatenCmd takes charge of when a *Character* object eats another *Treasure* object. *C2TCmdFac* is the factory for making *Character* to *Treasure* commands. *NullC2TCmd* ensures the robustness of the game.

5.3.5.6 SwitchStratCmd & UpdateStateCmd & NullCharacterCmd

SwitchStartCmd lets a *Character* object switch its current strategy. *UpdateStateCmd* updates the current status of a *Character* object. *NullCharacterCmd ensures the robustness of the game.*

5.3.5.7 AvoidStrategy & ChaseStrategy & RandomStrategy & RetreatStrategy & ScatterStrategy & NullStrategy

AvoidStrategy, ChaseStrategy, RandomStrategy, RetreatStrategy, and ScatterStrategy are all moving strategies of ghost objects. *NullStrategy* ensures the robustness of the game.

5.3.5.8 CollideStrategy & PacmanStrategy & StrategyFac

CollideStrategy determines the interaction between *Pacman* and *Treasure* objects. *PacmanStrategy* determines the behavior of Pacman. *StrategyFac* in the factory class for making different strategies.

5.3.5.9 Util

There will also be some utility classes to store parameters such as min dot numbers and initial live numbers.

Section 6 - Data Design

6.1 PacManStore

Team Team decided to use in-memory storage to store game data. Game maps will be pre-designed and hardcoded into the program.

Section 7 – References

- Course Rubric -- https://www.clear.rice.edu/comp504/#/assignments