

## Introduction:

\_\_\_\_\_ From a technical standpoint, Buck-Man is made up of three packages, “game”, “menu,” and “ghosts.”

The “game” package controls the actual levels of the game, and is broken down into the MVC classes PacmanModel, PacmanView, and PacmanControl, and also has the Enum classes Direction and CellValue.

The PacmanModel class reads a .txt file stored in resources to generate a maze with different maze features such as WALL, EMPTY, DOT, and BUCKYHOME, derived from the CellValue enum class. The model class is also responsible for moving Bucky and the ghosts through the cells of the maze. It keeps track of what object is stored in each cell and the coordinates of Bucky and the ghosts and updates the grid accordingly. It also uses the Direction enum class to store what direction Bucky and the ghosts are moving in, and change the ghosts’ direction when they come in contact with a wall. It also keeps track of the user’s score.

The PacmanView class takes the Model as an argument in order to access the map read in by the Model class and represents it with images stored in the resource file. It uses the coordinates for Bucky and the ghosts to keep the view updated. It also establishes the background image for the levels.

The PacmanControl class takes the model and the view as arguments and syncs their update methods to a Timer object. It also handles the user’s key pushes to determine the direction Bucky is travelling in. It also handles the InGameMenu that opens by pressing ESC, and pauses the game when it is opened.

The “ghosts” package contains only one class, the Ghost entity that contains the movement of the ghosts and attributes such as the X and Y coordinates of the ghosts and the direction they are moving in. It also has a method to handle changing the direction the ghost moves in, using the Direction Enum once again. It also stores the next coordinates the ghost will move into so that running into a wall can be anticipated by changing its direction.

The “menu” package contains 4 classes: MenuController, LevelController, InGameMenuController, and HelpMenuController.

The MenuController class accesses the MainMenu.fxml file stored in the resources folder to allow users to choose to start the game or quit. It also initiates the background music and gives users a chance to mute it if they wish. When the Start button is pressed, the scene is changed to the Level Select Menu.

The LevelController accesses LibraryLevel.fxml and allows the user to choose between the quad and library levels, thus opening the Game classes. Furthermore, users can press quit to exit the game, or Help to access the Help menu.

The InGameMenu class is triggered by the PacmanController class when the user presses Esc, and is stored as InGameMenu.fxml. PacmanController also pauses the game when this pop-up appears. Through this menu, users can access the Help menu, quit the game, or resume the game, causing the InGameMenu.fxml to close out.

The HelpMenuController class is the simplest menu, as it only displays HelpMenu.fxml and has a back button so users can return to whichever screen they were previously viewing.

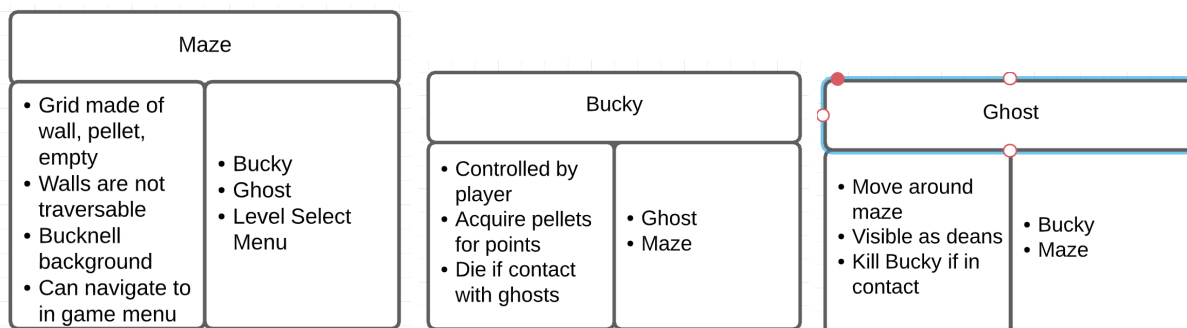
### User Stories:

User Story	Details
Bucky Sprite	Users can play as the visual representation of our Pac-Man character (Bucky the Bison)
Title Menu	Users can choose to start the game (leads to level selector menu) or quit
Level Selector	Users can choose different levels (locations on campus). Tentatively: Malesardi quad, Bertrand library
Ghost Sprites	Users are pursued by visual representation of ghost villains (potentially Bravman, Dean Badal, etc)
Quad Level Maze	Users can play on a level based off the Malesardi Quad
Library Level Maze	Users can play on a level based off the Bertrand Library
Bucky Sprite Movement	The user is able to control the sprite of Bucky with arrow keys or WASD
Ghost Movement	The users must avoid the ghosts, which will move around the levels
Maze Mechanics	The user is not able to move Bucky through the walls of the maze
Point Collecting	If user sprite (Bucky) runs into (eats) point sprites, it collects them and they are removed from the map
Point Total	User can see their points collected total in the top right of the game screen
High Score	Users can save their high score as their initials, if a new high score is achieved a new high score will be displayed on starting screen
Dying Mechanism	If Bucky sprite touches a ghost sprite, he dies and loses a life. If Bucky is out of lives, the game terminates
Showing Lives Remaining	Bucky starts with 3 lives displayed at the bottom. If he dies, one of these lives is removed
Losing the Game	If Bucky is out of lives, the game ends and a game over screen is displayed
In game Menu	Users can press ESC to bring up an in game menu that pauses the game and allows them to quit
Help Menu	Users can view a help menu with instructions
Music	Users can listen to and mute in-game music

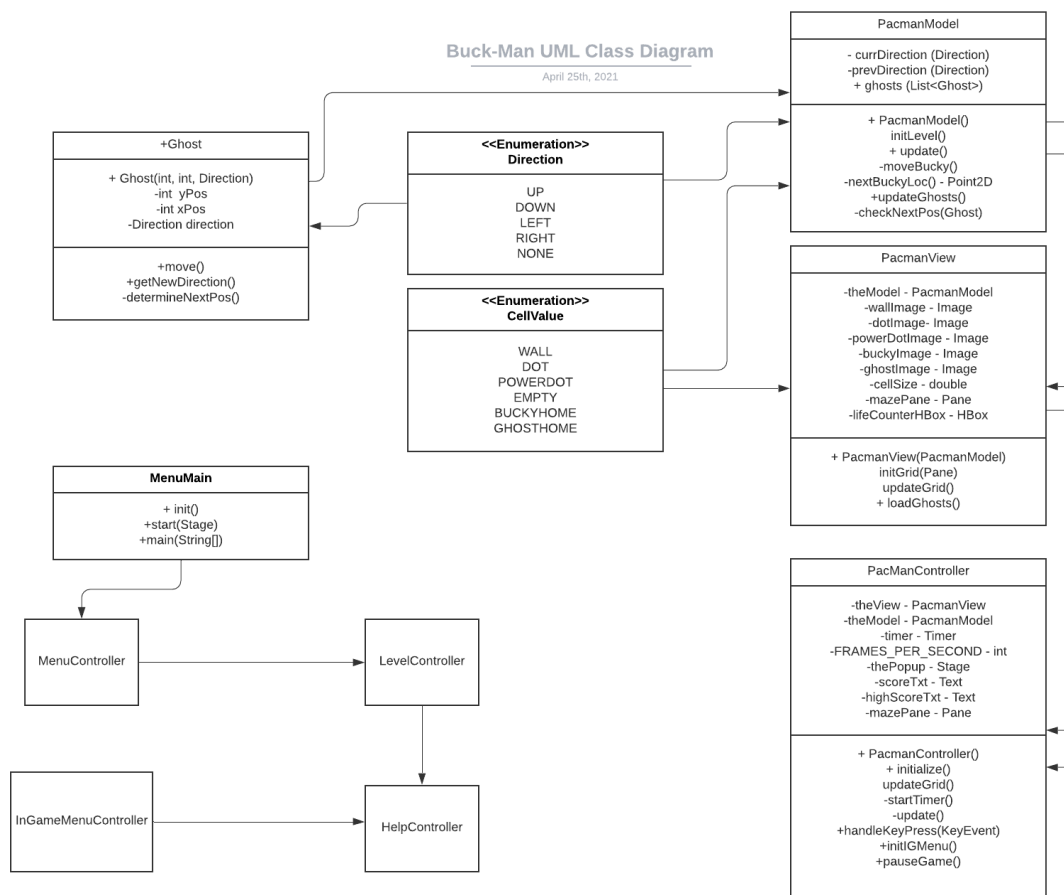
### OOD:

Our OOD ended up different from how was originally anticipated in our CRC cards. For instance, we initially envisioned a class for Bucky, and a class for the Maze (as seen in the CRC

cards below). This tied closely to our user stories, which separated the functionality of Bucky from the maze and the ghosts. However, we soon realized that it would be much easier to use MVC to create a Model, View, and Controller class. Therefore, the Model would be responsible for the maze and Bucky spatially, while the View would handle the visual representation, and the Controller would handle user input.

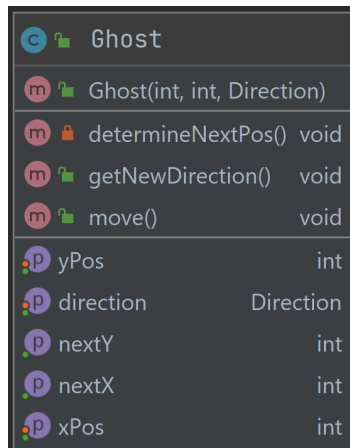


Therefore, our final OOD for the game appeared more like the UML diagram below:



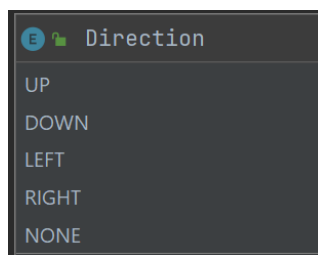
The Menu classes are less important to describe in the UML diagram, as they are fairly self-explanatory, but the way the MVC classes interact with each other and the Enum classes is shown through the arrows.

We chose to still make the ghost entities their own class because there are four ghosts that we wanted to behave similarly, and this made it easier to store the individual locations and directions of the ghosts without repeating code.



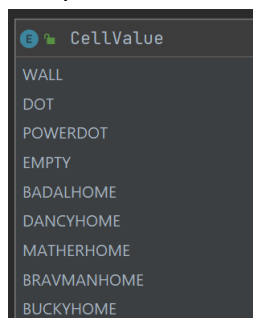
This screenshot from the IntelliJ generated UML diagram shows the ghosts more in-depth. Every ghost has its current X and Y coordinates stored, as well as its next X and Y coordinates, which are determined by its direction attribute. The `move()` and `getNewDirection()` methods are public because they are accessed by the other packages in `PacmanController`, but `determineNextPos()` is only used within the `Ghost` class (and is called by the two other functions in the class).

The directions of movement are stored in the `Direction` Enum class, which looks like:



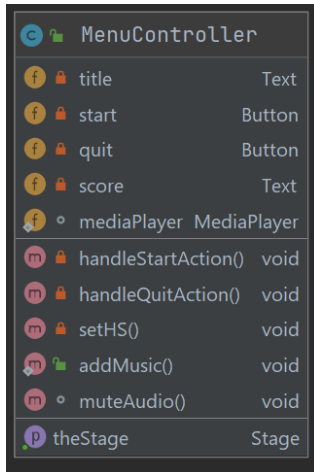
While the ghosts will only move in `UP`, `DOWN`, `LEFT`, and `RIGHT`, `NONE` is necessary for Bucky's movement prior to user input or when a wall is encountered.

The possible maze objects are stored in `CellValue`, another Enum class that looks like:



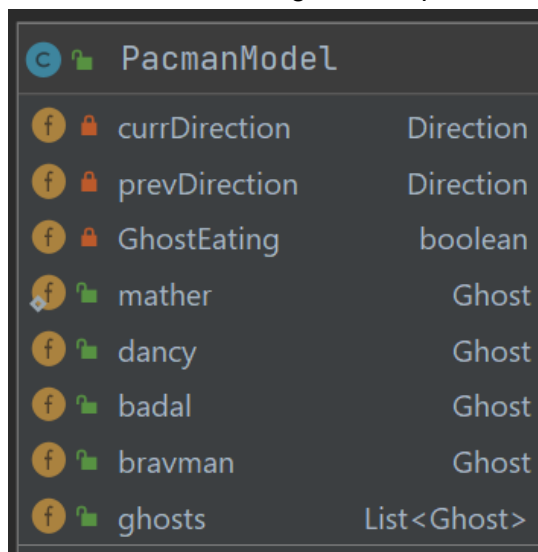
These are assigned by the PacmanController class, which reads a .txt file and assigns each cell a certain object. The PacmanView class then interprets these Enums to images stored in the resources folder.

All of the menu classes are very similar and contain mostly action handlers to lead to other menus.

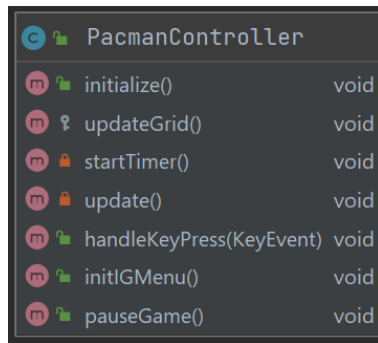


For example, in MenuController, the title menu, handleStartAction() is triggered by a mouse click on the Start button in the .fxml file, and then opens the LevelSelectMenu.

In the PacmanModel class, the ghosts are stored individually and in a list so that all of them can be moved at once using a for loop.



One of the parts of our OOD that we are most proud of is the use of a Timer in the PacmanController class that allows our model and view to be synced and updated together.



PacmanController		
m	initialize()	void
m	updateGrid()	void
m	startTimer()	void
m	update()	void
m	handleKeyPress(KeyEvent)	void
m	initIGMenu()	void
m	pauseGame()	void

Although there are not that many methods in the controller, it is responsible for the majority of gameplay. The timer also allows the ghosts to move independently at the same pace as Bucky, making the game challenging but still winnable.