Peg Puzzle Game Project Proposal

William Hayes, Jackson Horton

Abstract

Our project is a simple puzzle game where the goal is to remove as many pegs as you can from the board. The game ends whenever the player either 1) runs out of valid moves or 2) has only one peg remaining. Once the game is over, the program will display their score as well as a comment about their score like the infamous comments from the classic Cracker Barrel game. The target audience for the project is anyone with some spare time. Whether they are waiting in line at the store, passing some time before class starts, or bored at home, we hope to fill some of the void with an entertaining game. To achieve this, our project aims to be inexpensive, easy to use and keep our players occupied as long as they please. So far, we have achieved laying the foundations for the project's design and implementation, i.e. determining the requirements of our project. We hope to expand beyond the Cracker Barrel style, triangular board. This triangular peg game is an abstract variation of a classic peg game known as "Peg Solitaire" or "Hi-Q". As we develop the project, we want to keep future expansion in mind. A long-term goal of the project is to change what board the game is played on to accommodate the original board and its variations.

1. Introduction

This project is a simple puzzle game where the objective is to remove as many pegs from the board as possible until the player runs out of moves. Once the player has only one peg remaining, no valid moves to make, or decides to quit the game, the application will display the player's score (number of pegs remaining) along with some humorous text or commentary on a "game over" screen. Further, the program should check whether the player can make valid moves with the pegs after every move. The user will be able to move pegs by clicking a peg, then clicking an empty peg hole that is a valid move. The peg will be moved and remove the peg that is jumped over. When the user selects a peg they wish to move, all valid moves/peg holes will be highlighted green.

One of our goals is to implement mouse controls so the pegs can be dragged to the peg hole they wish to move to. If the peg is dropped at an invalid position on the board, it should return to its original position. If we implement this feature, the program should support both click movement and mouse drag movement.

We expect our target audience to be people of all ages. Anyone who need something to take up some of their time is our target audience. The program should function as a small piece of entertainment to keep users occupied and engaged. Furthermore, several features of the program, such as the "game over" screen, should function as conversational starters to assist in maintaining user engagement for as long as possible. In the Cracker Barrel version of the peg game, the board has titles for each number of pegs a player has left. Some are borderline insulting, but they are all funny and a part of the game that makes it unique and keeps you coming back. In short, the goal of the program is to entertain the user and keep them engaged.

1.1. Background

Terms:

- Peg: A piece on the board; is placed in a peg hole
- Peg hole: a hole in the board where pegs are placed; a peg hole can be empty or occupied (by a peg)
- Jump/Jumping: Moving one peg over another peg into a new peg hole; when a peg is jumped over, it is removed from the board and is no longer in play
- Board: the playing area which contains the peg holes where the remaining pegs reside
- Valid move: when one peg is jumped and another removed according to the rules outlined below

Rules:

1) The game takes place on a triangular board, with 15 holes for the pegs.
2) The game begins with pegs occupying all peg holes on the board except for one. Traditionally, the empty peg hole is one of the three central peg holes in the middle of the triangle.
3) To jump a peg, you must move one peg over an adjacent peg to the peg hole on the opposite side of the peg being jumped.
4) In order to jump a peg, it must jump into an empty peg hole.
5) If a peg is jumped over an adjacent peg, the adjacent peg that is jumped over is removed from the board.

6) The adjacent peg is not removed from the board if the peg hole directly opposite to the peg being moved is not empty.
7) If a peg has no adjacent pegs around it, it cannot jump or move until a peg is moved into an adjacent empty hole.
8) Progress in the game by jumping pegs.
9) The game is over when the player has no valid moves remaining. This occurs when there are no pegs adjacent to each other, or when a jump can not be performed on any adjacent pegs according to the other rules.

We choose the game for our project because we thought it would be a fun and challenging to implement a GUI and mouse input, and can show off software design patterns.

1.2. Impacts

Although we doubt that our program will have any serious social, economic, or cultural impact, there is an argument to be made that it is better for the environment than traditional peg games. Most of these puzzle games are made with wood or plastic boards and pegs; wood from chopped down trees, and plastics that eventually find their way into landfills. Our program is completely digital which helps avoid these environmental impacts. On the other hand, our program will require electricity which also has an impact, but is arguable far more efficient. Thus, there is a potential for the program to have a positive impact on the environment.

1.3. Challenges

There are many challenges that come with any graphical program, especially one that relies on where things are placed in the GUI. Our first challenge will be figuring our the best way to store a board with pieces in our program. This is a vital step in our program's development. Our next challenge will be the program's controls. This includes how the pegs are moves as well as menu navigation. We will then have to develop an efficient system for checking for valid moves, then use it to detect if the game is over and display the possible moves for any selected peg. A later challenge will be creating the leaderboard which will show who got the most pegs out in the quickest times and store this is a file so it can be loaded when the game is launched.

2. Scope

The scope of our project should be a fully functional peg game; i.e. the board and pegs are initialized in a new window, the peg pieces are movable with the mouse, the game checks the number of valid moves that can be made, there is a game over screen, and the game has some quality of life features implemented (like a quit button).

2.1. Requirements

The menu system was made a requirement as a quality of life feature. The menu includes operations like starting a new game, pausing the current game, quitting while in a game, and undoing the last move made by a user.

While in the main menu (not in a game), the program will display a "New Game" button and a "Quit" button. The "New Game" button will take them into the game screen which displays the board. The "Quit" button will close the program. The leaderboard should also be displayed on the main menu screen.

While playing the game, the program will show the board, all the remaining pegs, and the menu buttons. From the game, the user will be able to exit the game to the main menu with the "Exit" button and be able to reset their game to a fresh board with a "New Game" button. The program should also allow the user to undo the last move they made using the "Undo" button.

The last graphical requirement is the game over screen since it is a basic requirement in most other games. It will show the results from the game, like the time it took to finish and the number of pegs removed, as well as the "New Game" and "Quit" buttons.

Again the valid move checker is the most critical part of the program, it enforces the rules of the game. It is a core requirement for the program to function properly.

Stretch Goals:

- Pause button: add a pause button in the game that stops the timer and blurs the background so the user can take a break. It will also show a random fun fact that we will pull from a file.

- Timer: In the program's menu, next to the number of pegs left on the board, there should be a timer. The timer should start whenever the first peg is moved to be the user. The timer stops whenever the game is reset or the game over screen is reached. In the game over screen, the final time (in minutes and seconds) should be displayed below the number of pegs the user had on the board.
- Leader-board: On the main menu screen, the leader-board should display the top scores from the previous games. These scores are ranked by fewest pegs and least amount of time to complete, with fewest pegs having the higher priority. These scores should be stored in a file so the scores are persistent after the game is closed.
- New Boards: The game should allow the user to choose different game boards of various shapes and sizes to play on.

2.1.1. Functional.

- The program will display the board and pegs once the game is started and allow the user to make moves.
- Menu: The program, either at the top or bottom of the window, should have a menu. It should display the number of pegs on the board and buttons for the user to select, such as a quit button, an undo button, a help button, and "New Game" button.
- The program will allow the user to quit the game, closing the program.
- The program will be able to reset the game. This should not close the program, it should set the board and pegs to their original conditions.
- The program will show a "Help" button. This button should display a help screen that explains the rules of the game to the user and how to make moves.
- The program will calculate the valid moves. The game automatically checks the number of valid moves whenever a peg is moved. If it reaches zero, the game ends; otherwise, the game continues.
- The program will display a game over screen. Once the user reaches one peg or has no more valid moves to make, the program should display a game over screen. It should contain the final number of pegs left on the board, a humorous comment on the user's game-play, and the option to exit to the main menu or start a new game.
- The program will undo the last move made when the undo button is pressed.

2.1.2. Non:Functional.

- Performance: The program should be able to perform various functions and checks efficiently without major slow:downs that might hinder a user's experience. The valid move checker should be nearly instantaneous.
- Maintainability: The program should be organized to allow for easy readability and modification of code to quickly fix any bugs.
- Reliability: The program should be reliable with few bugs or performance issues.

2.2. Use Cases

Case 1: The user wants to quit the program from the main menu.

- Preconditions:
  1) The program is open and on the main menu screen.
  2) The user presses the quit button.

Case 1: The user wants to quit the program.

Precondition: The user has started a game and wants to quit the program.

1) The user moves the mouse to the Quit button and clicks on it.
2) In the message prompt that asks "Are you sure you want to quit?" the user clicks on the Yes button.
3) The user is redirected back to the program's main menu screen.
4) On the main menu screen, the user clicks on the Quit button.

Postcondition: The program terminates, if the leader-board stretch goal has been met then the user's information on the leader-board should be saved.

Case 2: The user wants to undo a move.

Preconditions: The user is currently in a game and has moved at least one peg.

1) The user clicks on the undo button.

Postconditions: The game resets the board conditions to be as it was before the last move was made.

Case 3: The user wants to pause the game.
Preconditions: The user has started a game, and moved at least one peg on the board.

1) User clicks on the Pause button

Postconditions: The GUI darkens slightly, displaying text on the screen that reads: “Paused”. The user can’t interact with the game board until they click on the resume button.

2.3. Interface Mockups

The initial interface is the main menu. On the main menu, the user can see the play and quit buttons, the leaderboard, and a yummy plate of chicken. The use cases involved with this screen are starting a game and quitting the game from the main menu. 1

Figure 1. Mockup of what the main menu will look like with the leaderboard.

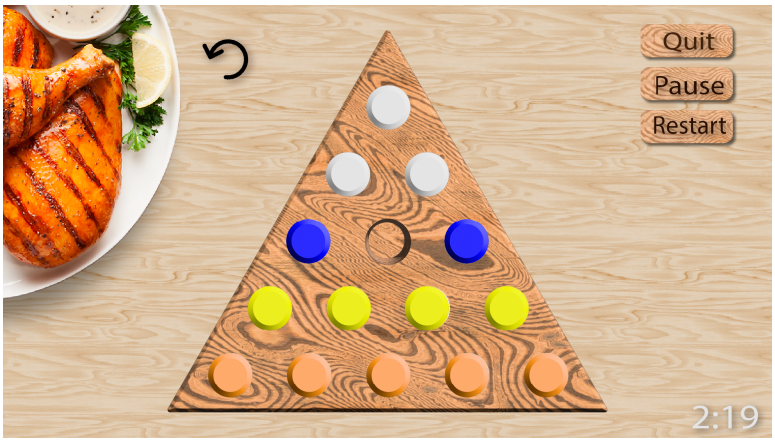

Figure 2. Mockup of the GUI during gameplay.

Figure 3. Mockup of the pause menu in the GUI.

3. Project Timeline

- Week 1 (January 8th)- Choose partners for the project.
- Week 2- Came up with idea from the project: creating a peg solitaire game.
- Week 3- Created the GitHub repository and basic requirements for the project.
- Week 4- Completed the project proposal draft and requirements.
- Week 5- Held project meeting to discuss possible changes and corrections to the draft.
- Week 6- We manage made corrections to the proposal draft and added in some use cases and interface mock-up.
- Week 7- We did plan a meeting to discuss possible design patterns and figure out how make menus on word using grid elements
- Week 8- Began the implementation stage by creating the hole and peg classes to serve as the project's framework.
- Week 9- Continue to work on project framework. Added in the menu and game board windows and the board class that uses the peg and hole objects.
- Week 10- Made the pegs movable on the board, and created the basic move checker algorithm.
- Week 11- Implemented the game over screen, factory method for button creation, and strategy pattern for the valid move algorithm,
- Week 12- Implemented the undo button, pause button, and the help buttons.

4. Project Structure

4.1. UML Outline

4.2. Design Patterns Used

- Factory Method Pattern- We used a factory method pattern to assist in the creation of button objects. This allowed us to easily add in buttons and their functionality to each window without making multiple mouse click event methods for each button,
- Strategy Pattern- We used a strategy pattern to allow for the creation and use of different valid move algorithms at runtime. We figured that when we'll implement different game boards to play on, we'll also need to use different valid move checking algorithms for those boards. So, using a strategy pattern would allow for the easy creation and usage of these different algorithms.

5. Results

We managed to create the peg and hole classes. Although we planned on the hole class having the most move functionality (by having this class use the actual valid move checker), we decided to put this logic in the peg class instead. As a side note, we originally planned on making a factory method for the peg and hole objects, but this was abandoned because we felt that

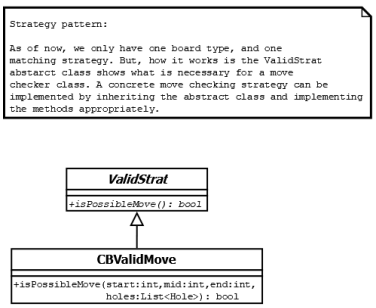

Figure 4. UML of the strategy pattern.

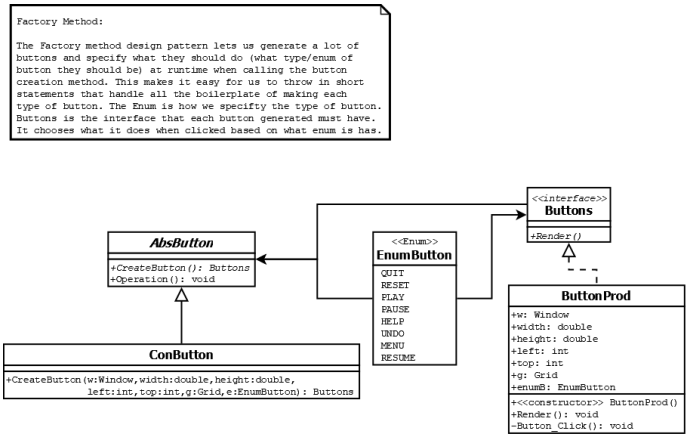

Figure 5. UML of the Factory Method pattern.

we were short on time and needed to quickly implement as much as we could. We also made the board window, which creates the actual game board and pieces. The window would later serve as the foundation of our project, with everything being slowly built off of it. After the game board was created, we implemented the main menu window. Originally we planned on creating windows programmatically; however, after a week of unsuccessful attempts, we abandoned the programmatic approach for the sake of time. We then implemented the actual logic for moving pegs around with the mouse, jumping and removing pegs, and the valid move checker. It was extremely difficult due to our lack of experience with WPF and C-sharp, but we were extremely pleased with the results. It is worth mentioning that we created the strategy pattern to allow us to follow the principle of ETC by allowing for the creation and usage of different valid move-checking algorithms. We also created the factory method to allow for the quick creation of the program's button UI elements. After this, we created the game over screen, adding a main menu, restart, and quit button. We also added logic that gives the user a little prompt that comments and playfully insults the user based on the number of pegs left on the board. Next, we created the undo button. Originally it was only going to undo one previous move, but after adding a list that stores moves that the user makes, we decided to have it undo any moves the user makes. Recently, we added functionality to the help and pause buttons and created their respective windows. For the pause menu, the difficult part was saving and somehow resuming the gameboard without changing the state of the board. However, we decided on having the pause button hide the game board window and having a resume button that unhides the game board window.

5.1. Future Work

So far we are on track, having implemented most of the basic requirements for the program. We managed to implement one of our stretch goals: the pause button. Right now our goals are to implement the stretch goals: a timer, then the leaderboard, and finally multiple gameboards. Once we're done with this project we plan on linking it to our resumes. If we desire to do so, we might return to it later to make improvements and turn it into a proper application.