Pegs & Dice

Final Report

Olivares Enthusiasts

Evan Delanty | David Sosa | Matt Benson

Course: CPSC 224 - Software Development

I. Introduction and Project Description

For the final project, we chose to create a GUI program of Pegs & Dice. Pegs & Dice is a strategic board game where players aim to move pegs into specific positions within a 5 x 12 grid by utilizing the outcomes of two dice rolls. Each player takes turns rolling the dice and strategically setting up combos to move their pegs to their upmost position. You can re-roll unused dice, but you can only use the re-rolls if it's the same result of the original combo. Pegs are moved forwards based on how many combinations of two dice equal the wanted result. The goal is to position all twelve pegs at the top level of the board in order to win!

This report summarizes our overall progress on the project and the technical details of our collective engineering efforts. The scope of this report is broad and includes topics such as our project requirements, solution approach, testing, implementation, and future goals.

II. Team Members - Bios and Project Roles

David Sosa, is a computer science student with an avid interest in computational finance and statistics. Some of his skills include Statistics, Python, and MW2 1v1s. These skills were used to their full potential during his internship last year at Marathon. For this project David was responsible for writing tests, and writing the overall logic for the program

---

Evan Delanty is a computer science student with a keen interest in software engineering and game development. Some general skills he has relate to programming in C++, Java, Python, and a little C#. These were used in school and personal projects done in Unity, a beginner friendly game development engine. For this project Evan was responsible for developing the interaction between the view and controller elements of Pegs & Dice, working hard to create a well designed and interactable user interface.

Matthew Benson is a computer science and mathematics student interested in inverse problems and medical imaging. His prior projects have included electrical impedance tomography (EIT) research. Matt's skills include C/C++, Python, Java, and MATLAB. For this project his responsibilities include developing non graphical class implementations, testing gameplay, and delivering positive vibes.

III. Project Requirements

This section includes the requirements for our Pegs and Dice game, More specifically, this section will cover major features, functional requirements, and non-functional requirements. These requirements guided the development of the game and served as the framework for our implementation.

Major Features

Feature	*Description*
Dice Rolling & Meld	A player must be able to simulate a dice roll and be able to choose a combination or number based on their inputs.
Board Status	A player must be able to see a live representation of their current board state and be able to see which pegs moved.
Re-rolling & Hot Hands	A player must be able to re-roll unused dice to try and create another combination of their previous combo(s) sum. Based on successful re-rolls they should get a new set of dice and continue their turn to find a different combination.
Winning	A player must be able to win if all their pegs have reached row 5.
Round Count	A player must be able to see what round it currently is (new round when all players have had their turn).

Functional Requirements

- A player must be able to simulate a dice roll and be able to choose a combination or number based on their inputs.
- A player must be able to see a live representation of their current board state and be able to see which pegs moved.
- A player must be able to reroll unused dice to try to create additional combinations with the same sum as their initial combo. If a player uses all of their dice or fills a column entirely, they should get a new set of dice and continue their turn with a new combination.

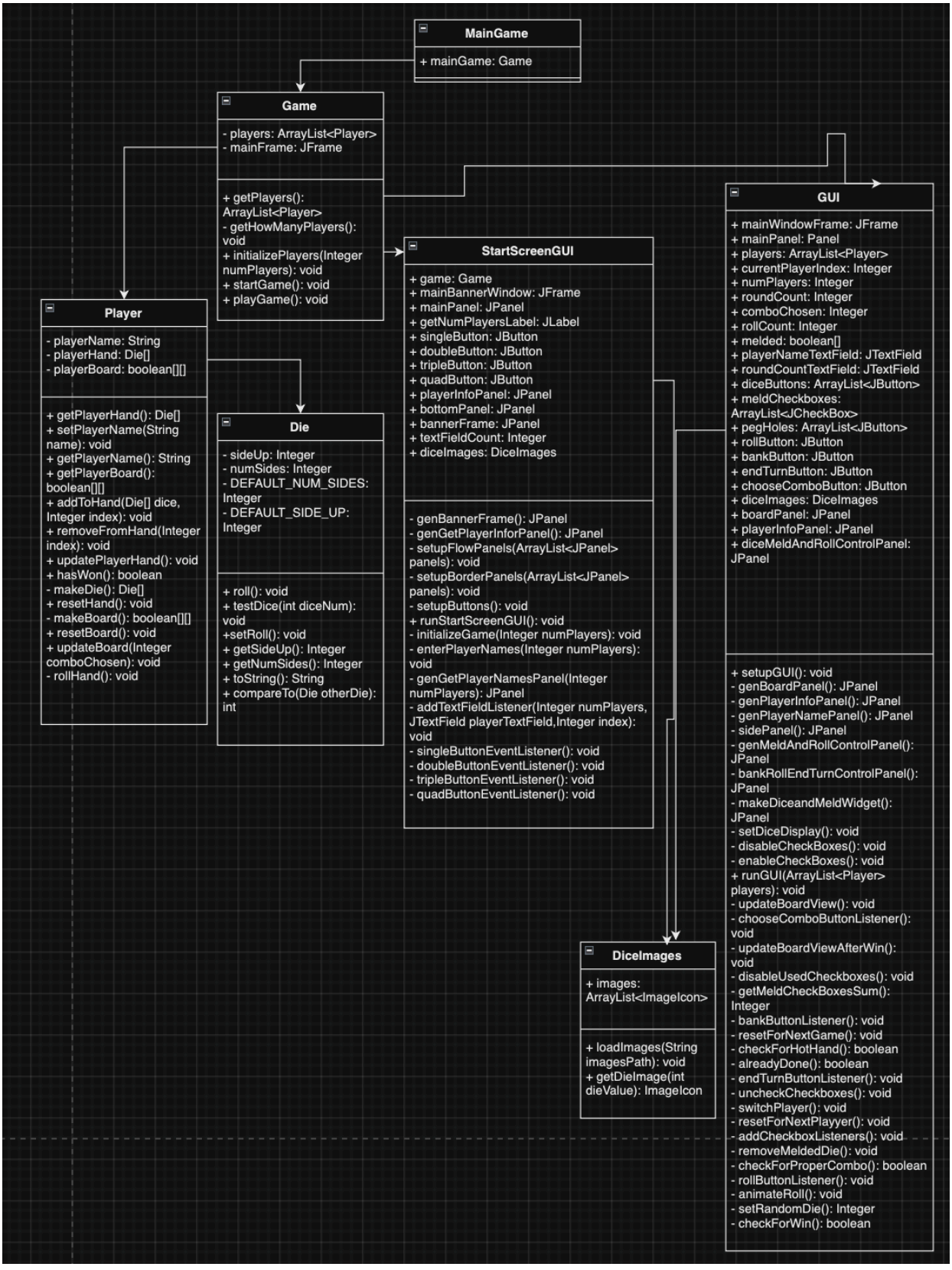
Non-functional Requirements

- A player should know exactly when a button is pressed from visual feedback.
- A player should be able to see the results of their input in under 5 seconds of system loading.

- A player should be able to visibly read all text on the screen no matter their screen resolution.

IV. Solution Approach

UML Diagram


MainGame
+ mainGame: Game
Game
- players: ArrayList<Player>
- mainFrame: JFrame
+ getPlayers(): ArrayList<Player>
- getHowManyPlayers(): void
+ initializePlayers(Integer numPlayers): void
+ startGame(): void
+ playGame(): void
GUI
+ mainWindowFrame: JFrame
+ mainPanel: Panel
+ players: ArrayList<Player>
+ currentPlayerIndex: Integer
+ numPlayers: Integer
+ roundCount: Integer
+ comboChosen: Integer
+ rollCount: Integer
+ melded: boolean[]
+ playerNameTextField: JTextField
+ roundCountTextField: JTextField
+ diceButtons: ArrayList<JButton>
+ meldCheckboxes: ArrayList<JCheckBox>
+ pegHoles: ArrayList<JButton>
+ rollButton: JButton
+ bankButton: JButton
+ endTurnButton: JButton
+ chooseComboButton: JButton
+ diceImages: DiceImages
+ boardPanel: JPanel
+ playerInfoPanel: JPanel
+ diceMeldAndRollControlPanel: JPanel
+ setupGUI(): void
- genBoardPanel(): JPanel
- genPlayerInfoPanel(): JPanel
- genPlayerNamePanel(): JPanel
- sidePanel(): JPanel
- genMeldAndRollControlPanel(): JPanel
- bankRollEndTurnControlPanel(): JPanel
- makeDiceandMeldWidget(): JPanel
- setDiceDisplay(): void
- disableCheckBoxes(): void
- enableCheckBoxes(): void
+ runGUI(ArrayList<Player> players): void
- updateBoardView(): void
- chooseComboButtonListener(): void
- updateBoardViewAfterWin(): void
- disableUsedCheckboxes(): void
- getMeldCheckBoxesSum(): Integer
- bankButtonListener(): void
- resetForNextGame(): void
- checkForHotHand(): boolean
- alreadyDone(): boolean
- endTurnButtonListener(): void
- uncheckCheckboxes(): void
- switchPlayer(): void
- resetForNextPlayyer(): void
- addCheckboxListeners(): void
- removeMeldedDie(): void
- checkForProperCombo(): boolean
- rollButtonListener(): void
- animateRoll(): void
- setRandomDie(): Integer
- checkForWin(): boolean
StartScreenGUI
+ game: Game
+ mainBannerWindow: JFrame
+ mainPanel: JPanel
+ getNumPlayersLabel: JLabel
+ singleButton: JButton
+ doubleButton: JButton
+ tripleButton: JButton
+ quadButton: JButton
+ playerInfoPanel: JPanel
+ bottomPanel: JPanel
+ bannerFrame: JPanel
+ textFieldCount: Integer
+ diceImages: DiceImages
- genBannerFrame(): JPanel
- genGetPlayerInforPanel(): JPanel
- setupFlowPanels(ArrayList<JPanel> panels): void
- setupBorderPanels(ArrayList<JPanel> panels): void
- setupButtons(): void
+ runStartScreenGUI(): void
- initializeGame(Integer numPlayers): void
- enterPlayerNames(Integer numPlayers): void
- genGetPlayerNamesPanel(Integer numPlayers): JPanel
- addTextFieldListener(Integer numPlayers, JTextField playerTextField,Integer index): void
- singleButtonEventListener(): void
- doubleButtonEventListener(): void
- tripleButtonEventListener(): void
- quadButtonEventListener(): void
Player
- playerName: String
- playerHand: Die[]
- playerBoard: boolean[][]
+ getPlayerHand(): Die[]
+ setPlayerName(String name): void
+ getPlayerName(): String
+ getPlayerBoard(): boolean[][]
+ addToHand(Die[] dice, Integer index): void
+ removeFromHand(Integer index): void
+ updatePlayerHand(): void
+ hasWon(): boolean
- makeDie(): Die[]
+ resetHand(): void
- makeBoard(): boolean[][]
+ resetBoard(): void
+ updateBoard(Integer comboChosen): void
- rollHand(): void
Die
- sideUp: Integer
- numSides: Integer
- DEFAULT_NUM_SIDES: Integer
- DEFAULT_SIDE_UP: Integer
+ roll(): void
+ testDice(int diceNum): void
+setRoll(): void
+ getSideUp(): Integer
+ getNumSides(): Integer
+ toString(): String
+ compareTo(Die otherDie): int
DiceImages
+ images: ArrayList<ImageIcon>
+ loadImages(String imagesPath): void
+ getDieImage(int dieValue): ImageIcon

V. Test Plan

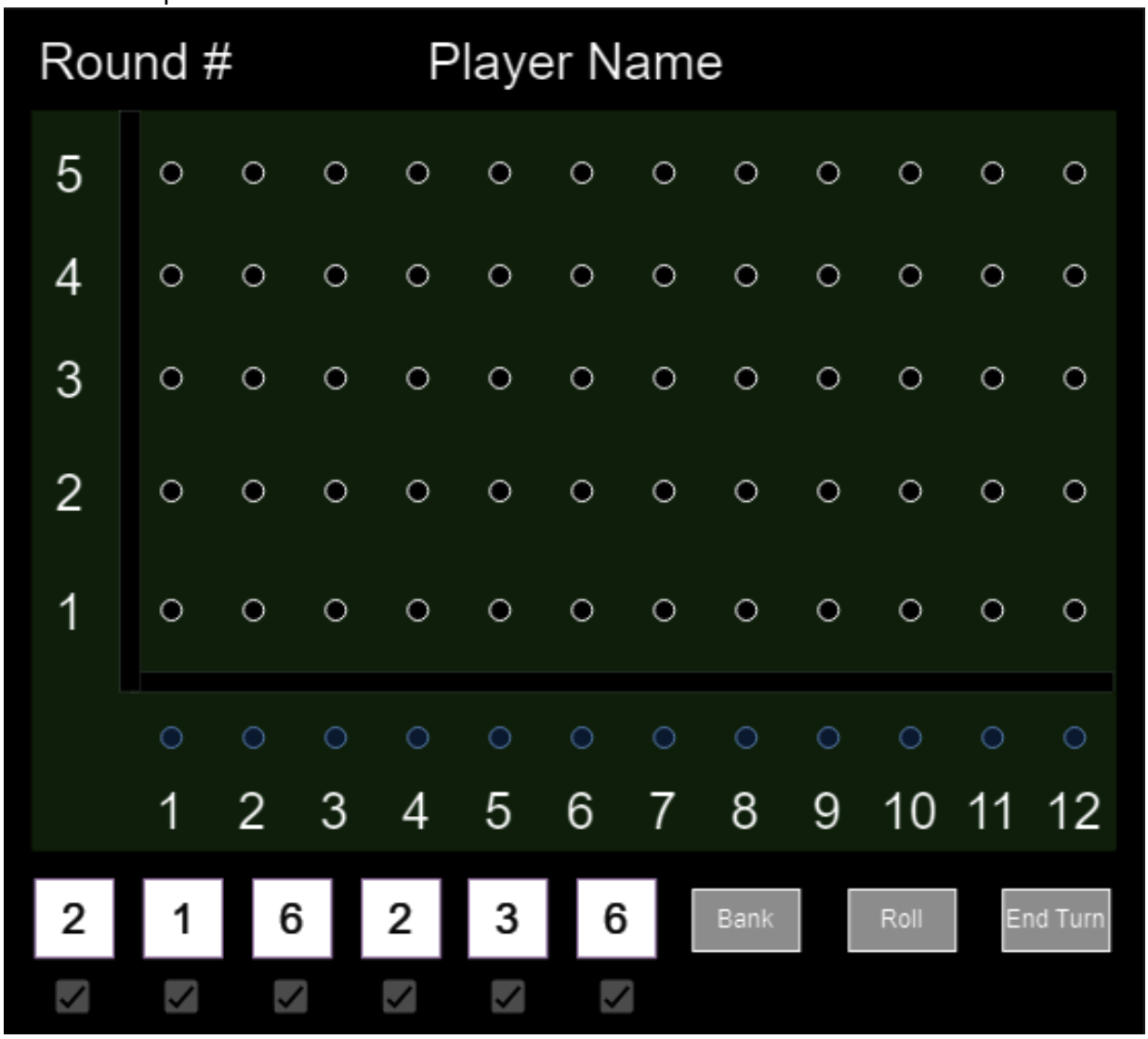
Our testing consisted mostly of a combination of unit tests and user tests. As our program utilized a GUI, we relied heavily on user tests. Our unit tests focused on foundational aspects of our program such as determining if a player has won, determining whether a player's board correctly was correctly reset, determining whether a player's hand was correctly reset, and determining if a player's board correctly updated in response to a player melding their dice. With the knowledge that these core aspects of the program maintained their functionality even as we made large changes to the code, we were able to identify any potential failures or defects in our program and their sources. Moreover, our unit test for determining if a player has won was essential as reaching a winning game state requires far too much time for user tests alone.

Through the use of GUI tests, we were able to verify that features such as rolling/rerolling dice, melding dice, receiving a new set of dice for hot hands, displaying an accurate round count, and maintaining an accurate representation of the player's current board state worked as intended. Moreover, we had our friend who created the game and others interested in our project play through the game. These user tests were essential to detecting issues and to ensuring that our non-functional requirements were met. The feedback from these tests gave us confidence in the speed, responsiveness, and visibility of our program.

VI. Project Implementation Description

Git repo link:
https://github.com/GU-2024-Spring-CPSC224/final-team-project-olivares-enthusiasts

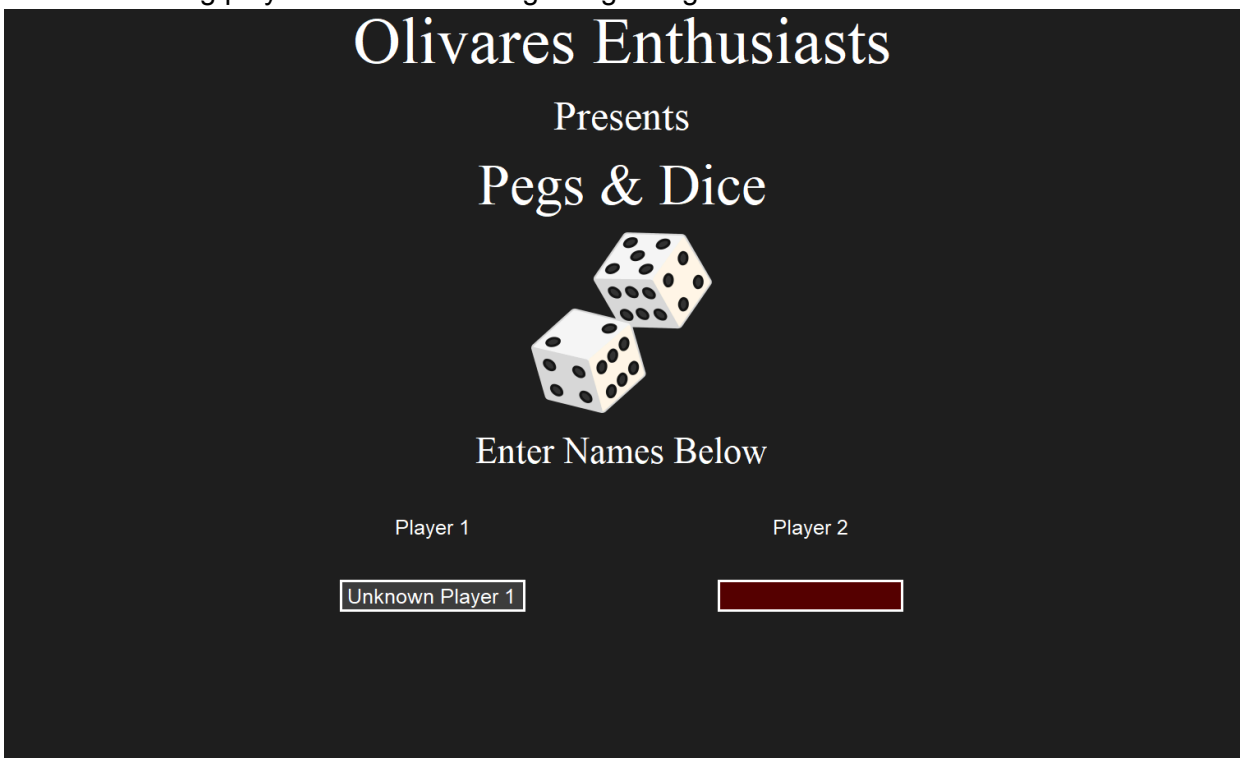
Initial Mock Up:



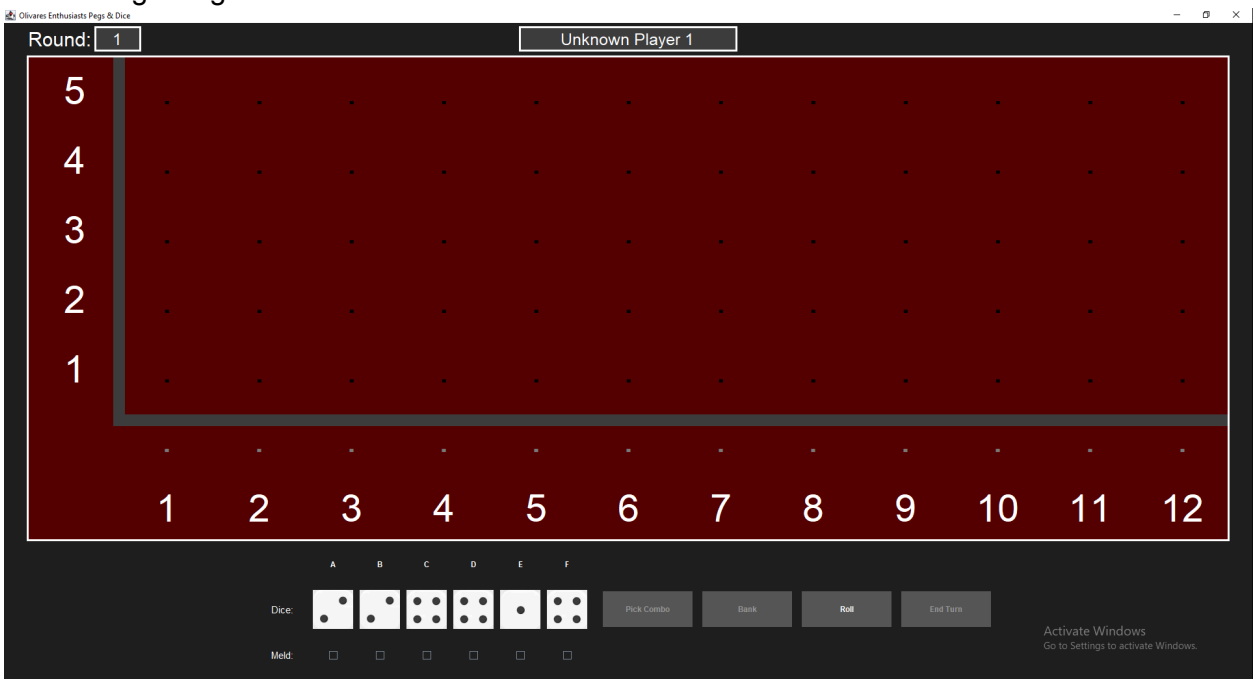
While we still have room for improvement, our team effectively implemented the entirety of our proposed functionality. The game runs as we envisioned from our initial planning stage, and our project has outdone the appearance of our initial mock up.

Program start screen with the option of playing either singleplayer or multiplayer:

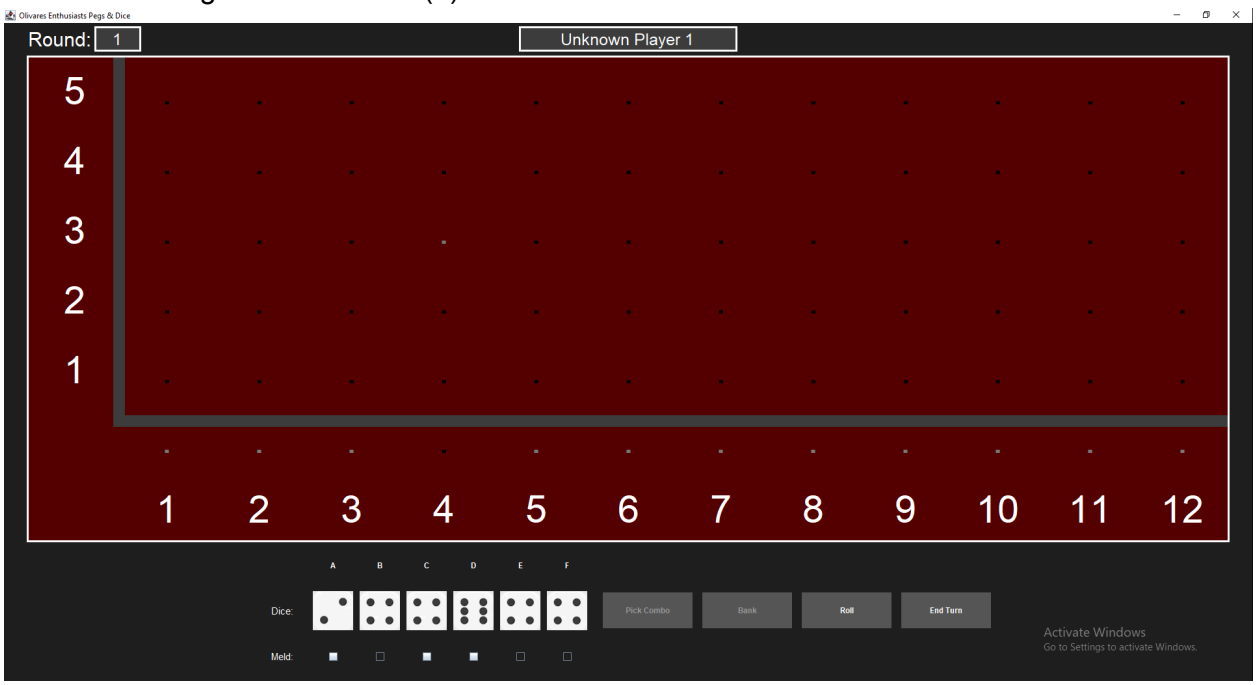
Selecting player names at the beginning of a game:



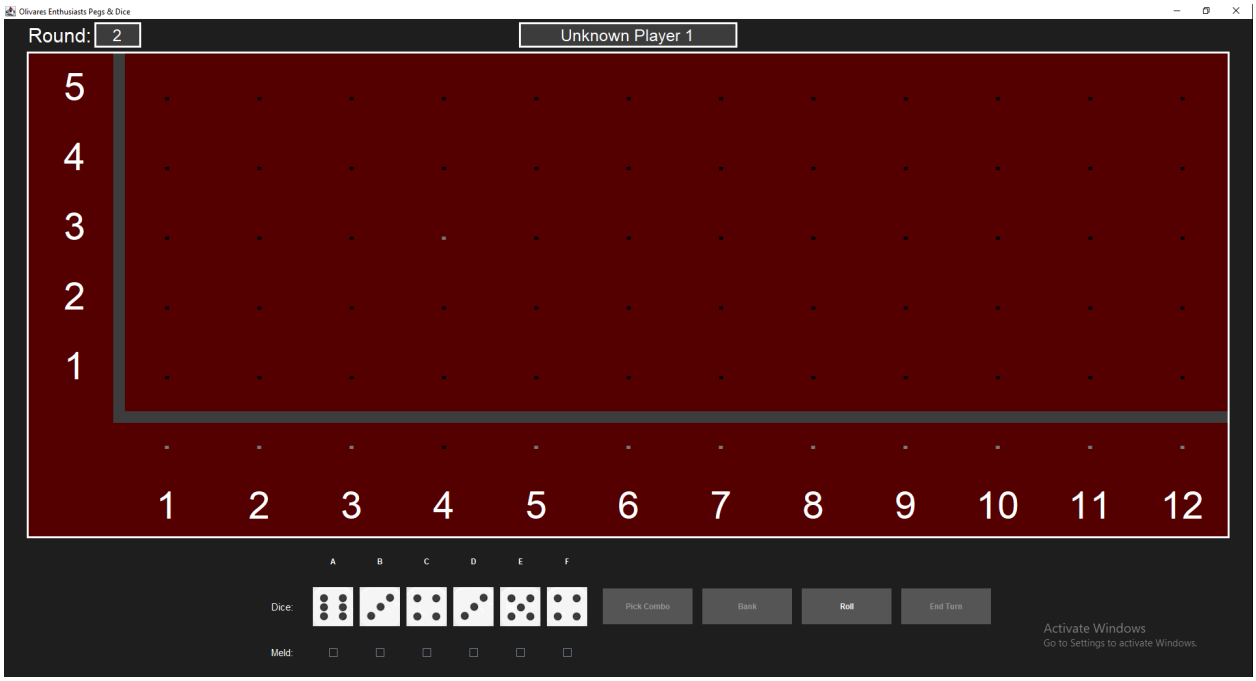
Beginning of a round:



Selecting a combo value (4):



After a round:



VII. Future Work

For the most part, we are very satisfied with our game. However, a feature that we want to implement in the future is customizing the appearance of the board and pegs. We think that adding in options for custom peg icons and varying board colors would lead to a more enjoyable experience and further differentiate each board in a multiplayer game.

VIII. Glossary

Definitions:

- Unit Tests:
  - Unit tests are used to verify that the smallest units of code work as intended.
- UML:
  - Unified Modeling Language