Title: Bubble Trouble

Group members: Isabel D'Alessandro, Leslie Gates, Michelle Quin

User's Manual

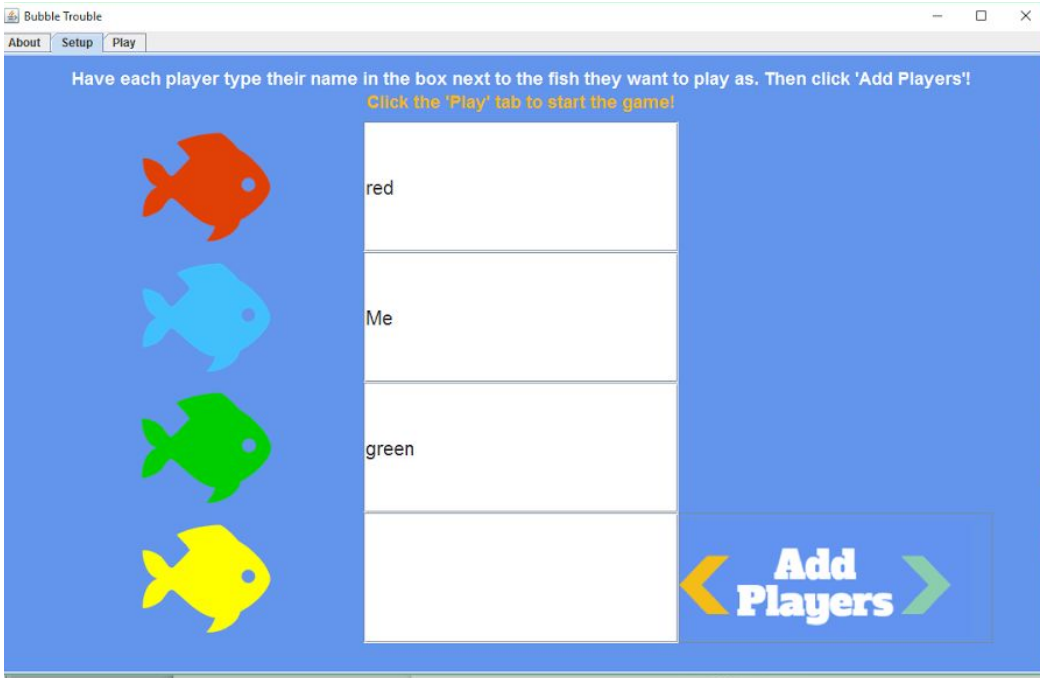
Description: We are making the board game Trouble, a 2-4 player game with a grid-like board, a dice, and pawns. The goal of the game is to be the first player to move all four of their pawns from "Home" to "Finish." A player rolls the dice; the dice must land on 6 or a 1 to initialize a pawn in play (from "Home" to "Start"). Once one or more pieces has been moved from Home to Start, you move your pieces around the board clockwise based on the number rolled on your turn.If you are able to land on a space that has an opponent's pawn, your opponent's pawn is sent back to their "Home." You may not land on your own pieces, including to move onto "Start." Once the pawn has made a full loop around the board, it can be moved into the "Finish". Once somebody has all four pawns in the "Finish", they win and the game is over.

GUI:

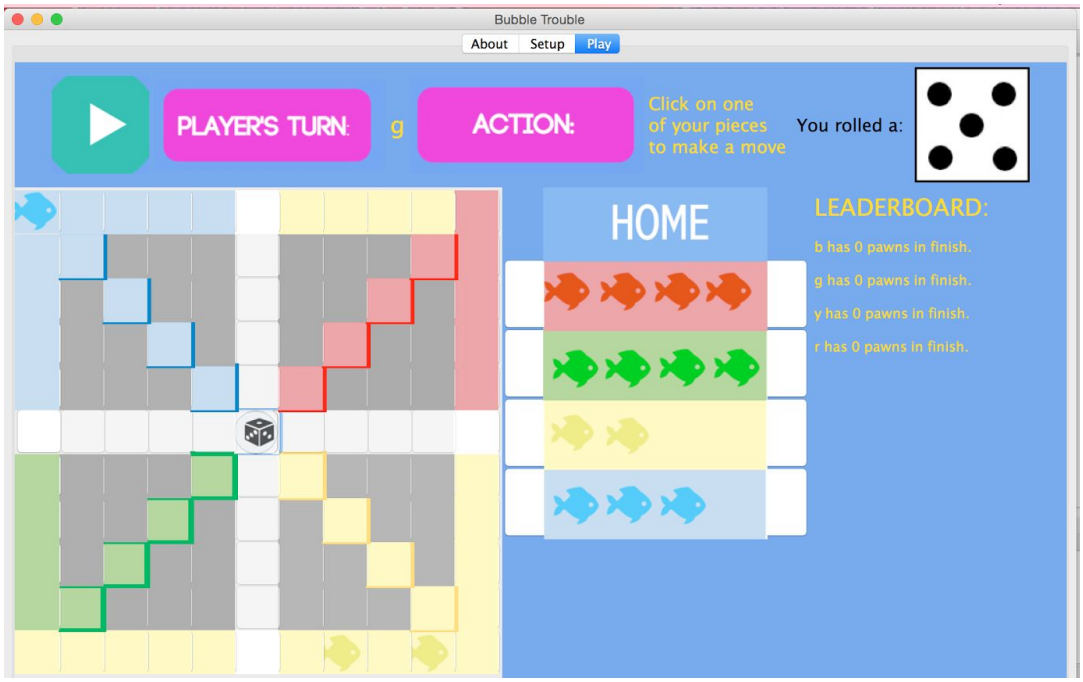
There will be 3 tabs in the GUI: Instructions, Setup, and Game:

The instructions pane will have all of the instructions needed to play the game and a picture of the game board.

The setup pane will allow players to pick their game pieces and their name. A Player will enter their name in the text field next to the color/icon that they would like to be in the game. If there are fewer than 2 players' names entered, they will automatically play a computer player.


Bubble Trouble
About
Setup
Play
Have each player type their name in the box next to the fish they want to play as. Then click 'Add Players'!
Click the 'Play' tab to start the game!
red
Me
green
Add Players

The Game tab is where the actual game will be played. On the top of the Game tab is a label indicating which Player's turn it is, and what action they need to take(roll, move, etc.). To the right of the game board will be a panel displaying which Player is currently winning. The game starts by clicking on the play button in the upper left corner. Once that button is clicked, the first player will be prompted to roll the die by clicking on the die icon in the center of the board. The leaderboard will also be initialized at this point. After rolling, the number that was rolled will be displayed in the upper right corner. It will then prompt the next player to roll if the current player cannot make any moves or will prompt the current player to select a pawn to move.



Interactions during gameplay:

(A randomly generated gameplay order will determine which player goes first, with the rest of the players going in clockwise order.)

For general gameplay:

There will be a message on the top indicating whose turn it is and what action should be taken(roll, select a piece to move, etc.). To roll, the player will click on the die icon which will randomly switch to a different side/number and display this number to the player. A player can move one of their pawns by clicking on an available pawn icon. The pawn will automatically be moved the number of spots rolled. They may only click on a pawn in their home if they roll a 6 or a 1, if they click on it and they haven't rolled a 6 nothing will happen. If they click on any space that does not contain one of their pawns, nothing will happen. Once a player has completed their move, the game will check to see if there is a winner. If there is a winner, a message will pop up declaring the winner. Otherwise, the message indicating whose turn it is will change and will prompt the next player to roll.

Technical Report

Data Structures: GUIs, Queue, Max Heap, Array

GUI:

This will be used to display the instructions and setup as well as for the game play.

Max Heap:

This will be used to determine which player is winning at any given point in the game. This information will be displayed in a panel to the right of the game board. The winning player will be determined by the player with the greatest number of their pieces in the finish zone. If multiple players have the same number of their pieces in the finish zone, they will be displayed one after another. Ties are possible. We chose to use a max heap because it will provide us with an easy and efficient way to access the maximum element, in this case the player with the maximum score. It is a dynamic data structure, so this information can be updated as game play continues. We could also potentially use order statistics with the heap to identify the player in 2nd, 3rd place, etc. at a given moment.

Array:

We will use an array of Pawn objects to represent the actual spots in the game board. When a pawn is present in a given spot on the game board, represented by an index in the array, it will be stored in that index in the array. When the pawn is to be moved, the Pawn object will be stored in the new index, and the value of the old index will be set to null. We will also use smaller arrays of Pawns to represent the "home" regions and the finish zones for each player.

Queue:

The queue will hold the players' names once the order of play has been determined. This allows up to retain the order of the players by enqueuing the current player once their turn is done and dequeueing the next player for their turn. There are always four players in the queue. If only one Player is added using the 'Setup' pane, ComputerPlayers will automatically be added to the queue until there are 4 players.

Classes:

Die():
From a previous assignment, it will return a random number between 1-6. Contains the roll method.

Player(color):
The player class has 4 instance variables: Die, color, name and an array of 4 pawns.
There are two constructors, if no name is given, then the color is assigned to be the name, otherwise they are the same. Both assign the 4 pawns to the color given. A Player has the ability to roll the die when they click on the die button. They can also move pawns by clicking on them. The class also keeps track of how many pawns are in play at that moment, home many of the player's pawns are in the finish zone, and whether the player is winning at the current moment.

ComputerPlayer(color):
This class creates a ComputerPlayer object which inherits from the Player class. It automatically rolls when it is its turn and will move a pawn out of home into start if possible. Otherwise, it makes a random possible move (this automatic determination will occur in PlayGame class). This is for computer players occurs only for computer players.

Pawn (color, position):
Each pawn has 5 instance variables: its color, position as well as its start and finish position, and a boolean variable canFinish, which indicates whether the pawn is capable of moving into the finish zone.These positions are calculated through the color index in the array [blue, red, yellow, green], where blue's start position is in the array at place 1 (since to calculate the finish position I subtract 1). All others increase by 10 to get the starting position.

Board():
This creates a Board object, which lays out the game board using the array. It is used to keep track of where pawns are and to move pawns around the board.

PlayGame ():
This class is responsible for initiating the game once players have been chosen. It invokes the classes Board, Pawns, Player, ComputerPlayer and Die, and the methods move(), turn(), win(); isComputer(),getBoard(), setupPlayers(String p1, String p2, String p3, String p4), orderPlay(String p1, String p2, String p3, String p4),nextPlayer(), trouble(), winning(), isWinner()

GUI InfoPanel():
Class that creates the first tab that you see when you launch the game. It has the title of the game as well as the instructions on how to play the game with a underwater background.

GUI SetupPanel(PlayGame game):

Class that creates setup panel for the game. It inherits the game from the BubbleTroubleGUI. It holds textfields so that players can enter their names and select which color they would like to play. It invokes a PlayGame class method orderPlay() on the game to create the players and pawns. This also allows the players to be accessible to the PlayPanel.

GUI PlayPanel(PlayGame game):
The panel where the main game is played. It displays the board, the 4 home zones containing the players' pawns, a leaderboard indicating which player is winning based on the number of pawns they have in the finish zone, as well as labels displaying whose turn it is and what action they should take.

GUI BubbleTroubleGUI():
The driver for the game. Creates the 3 tabs of instructions, setup and game. Contains an instance of PlayGame that is inherited by SetupPanel and PlayPanel.

Methods:

Die class:
roll(): int, returns a number 1 thru 6

Board class:
checkPosition(int position): boolean, *helper function*, checks to see if there is a pawn in that position of the board
getBoard(): Pawn[ ], returns the pawn array that contains all the pawn on the board and their positions
getPawn(int position): Pawn, uses checkPosition(int) to see if there is a pawn in that position, if there is one, it returns that pawn (useful for the trouble() method)
setPosition(int position, Pawn p): void, sets a position to have a Pawn or a null (for moving pawns around the board)
checkPosition(int position) boolean, helper function to determine if there is a pawn in a given location on the board

Pawn class:
isColor(String color): boolean, *helper function,* checks to see if the color is valid (prints out a message if not)
sPosition(String color): int, *helper function,* calculates the start position of the pawn based off its color
getColor(): String, gets the color of the pawn
getPosition(): int, gets the current position of the pawn, returns -1 if in the homezone and -2 if in the finish zone
setPosition(int position): void, allows you to set the position of the pawn, limited from -2 to 41. If the position is >= 42 it takes the remained when that position/42 and sets the canFinish to true. (which allows you to say that if the pawn position>finish position then set its position to -2)
getStartPosition(): int, gets the starting position for the pawn, varies by the color
getFinishPosition(): int, gets the finish position for the pawn, varies by the color

getCanFinish(): boolean, returns true if the pawn can finish and false if it can't
setCanFinish(): void, sets canFinish to false (for use when pawn is sent back to home)
toString(): string, returns the string "The " + color + " pawn is at position" + position

Player class:
getColor(): String, returns the color of the player
getPawnArray(): Pawn[ ], returns the 4 pawns the player has
getName(): String, returns the name of the player
getRolled() boolean; getter method for the rolled variable indicating whether the player has already rolled the die
rollDie(): void, rolls the die that returns a number from 1 to 6 and sets it to an instance variable
getDieNumber(): int, gets the number from the instance variable die
inPlay(): int, returns the number of pawns that are on the board which excludes the home and finish zones, (0-41 indicates that the pawn in on the board)
inFinish(): int, returns the number of pawns in the finishzone as determined by the pawns position (-2 indicates that the pawn is in the finish zone)
inHome(): int, returns the number of pawns in the homezone as determined by the pawns position (-1 indicates that the pawn is in the homezone)
oneHome() Pawn, returns a pawn that is currently in the home zone
noMoves(): boolean, returns false if there is no valid move and true if there is a valid move.
compareTo(Player p2): int, compares the number of pawns in the finishzone between the two players
getHasMoved() boolean: getter method for the hasMoved variable which indicates whether the player has already moved during their turn
setHasMoved() void: setter method for the hasMoved variable
compareTo(Player p2): int: compares the number of pawns in the finish zones between 2 players; used to determine which player is winning for use in the max heap/leaderboard
toString(): String, returns the name of the player plus the number of pawns in the finishzone

ComputerPlayer class:
pawnIndex(): int[ ], *helper function*, returns an array with the indices of the pawns in the pawn array that are in play(NOT their position, but pawn 0 in play or pawn 3 is in play ect)
startOccupied(): boolean, *helper function*, checks to see if there is a pawn in the start position
pickPawn(): Pawn, returns the pawn that should be moved

- If there is a pawn in the start position, returns the index of that pawn
- If not:
    - Returns an index of a pawn at home if you rolled a 6 & there is a pawn in home
    - If only 1 pawn in play, choses that one
    - Otherwise, choose a pawn that has canFinish==true
    - Else, randomly select a pawn
    - Doesn't need to worry about the case where there is no valid move as the Player method noMove() is checked beforehand

PlayGame class:
isComputer(): boolean, *helper function*, checks to see if the player is a computer player, given that an empty string indicates a computer player

setupPlayers(String, String, String, String): void, *helper function*, creates the 4 players given 4 strings. If the string is empty creates a computer player with the color as its name
orderPlay(String, String, String, String): LinkedQueue<Player>, *helper function*, takes in 4 strings of the players names. Calls upon setupPlayers() to create the 4 players. Automatically rolls for each of the players and then figures out the order by the highest rolling player.
getBoard(): Board, *helper function*, returns the board object
nextPlayer(Player p): boolean, checks to see if the player can move, if they can leaves the player alone. If they don't have any valid moves, enqueues the current player and returns true. Returns true if there is a valid move, false if not (false is given if they didn't roll a 6 and they have no pawns in play)
move (Player, pawn): void, first checks to see if the player can actually move, if they can't goes on to the next player. Then it sets the pawn's current position to null and moves it to its new position based on what was rolled.

- If the pawn is at home and the player rolled a 6, its position is moved to the start position so long as there is no other pawn of that player in the start position
- If the pawn is in play, then it moves forwards the amount as shown on the die
- If the pawn reaches the finishzone, it moves into the finishzone

If in any of these moves, the pawn lands on another player's pawn, the method trouble is invoked on that pawn and that pawn is sent to home. The method ends by calling the nextPlayer() method. (MAY BE AN ISSUE BECAUSE WE NEED TO MOVE ON TO THE NEXT PLAYER, but nextPlayer only checks to see if they have a valid move, may need to manually enqueue and dequeue the player in this case)

trouble(Pawn): void, takes in a pawn and sends it back to home, ensuring that its previous position is set to null again and that its canFinish variable is set to false
winning(): String, uses a priority queue to figure out the order of who is winning based off how many pawns they have in the finishzone.
isWinner(): boolean, checks to see if any of the players have all 4 pawns in the finishzone, if one does returns true, else returns false.

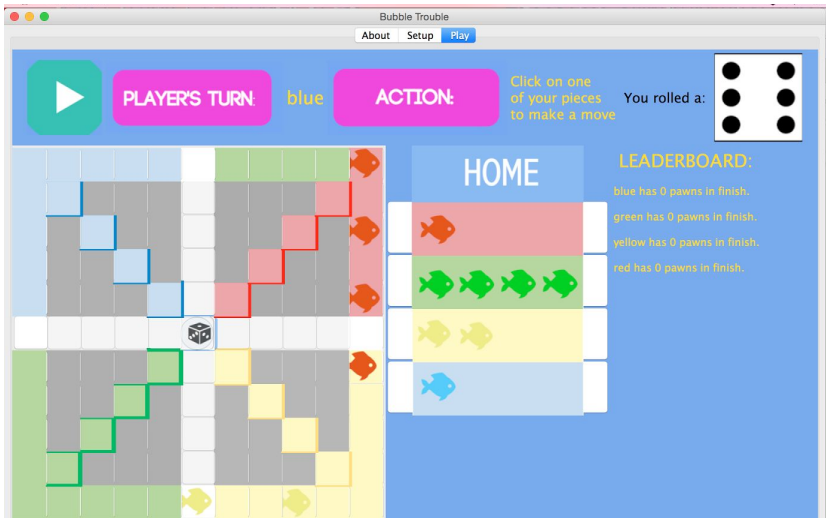
SetupPanel: GUI
createImageIcon(String path, String description): Icon, *helper function*, allows for the creation of an icon/picture
ButtonListener class: when the addPlayers button is clicked, it creates all the players based on whether they have a name using the orderPlay method

PlayPanel: GUI
makeCenterPanel() JPanel: helper function, which creates a center panel for the PlayPanel using the grid layout . This includes the board that is clickable
createImageIcon(String path, String description) ImageIcon: helper function which creates an icon that can be added to the panel; used to create all ImageIcons used in the GUI
getBoardPosition(JButton button) int; helper method which takes in a JButton and returns its position in the Pawn array on the board; used for finding the corresponding image in the array based on a Pawn's position in the GUI
pawnEnabled(Player p) void: helper method that sets occupied pawn positions on the board to enabled, so that the player can click on them after rolling

boardEnabled(boolean s): void: helper method which enables or disables the buttons that serve as the path that the pawns travel on. It is used to help indicate to the player what they should be clicking on
getGUIPosition(int boardindex) JButton: helper method which takes in an integer, which is the index of a Pawn in the array representing the board. It returns the corresponding position on the grid of button. This method is used for finding where the image should be changed in the GUI based on the pawns new position after moving.
getColorZone(int boardindex) String: takes in an integer that is the board index of a pawn found from the PlayGame class, and returns a string indicating which quadrant it is in on the board represented in the GUI(red, blue,green,yellow). It is used to determine which image to choose to represent the new position of a pawn after moving.
isValidSelection(Player p, JButton button) boolean: takes in a player and the button that the player clicked on and checks to see that it is a valid selection. If the button is considered a finishzone, then it is not a valid selection. If the player clicks on another players pawn then it is not a valid selection If the player selects their own homezone, returns true, otherwise returns false.
chooseHomeImage(Player p) ImageIcon: takes in a player and updates the player's homezone. For each player, there are 5 images: 4 pawns, 3 pawns, 2 pawns, 1 pawn and no pawns in the homezone. The method checks to see which player it has and then updates the appropriate homezone.
updateBoard() void: sets the proper icons for all positions on the board and home zones based on the status of pawns in the Board array. It is used to reset the board after evry turn when revalidating/repainting
chooseBoardImage(String color, int boardindex) ImageIcon: takes in a player and an integer that is the board index and returns the appropriate image for the pawn.It first checks what color the player is and then finds out which zone the pawns will be going to. Based off what color zone the pawn will be in, it has a unique icon that will be placed at that index in another method.
actionPerformed(ActionEvent event) void: Determines out what button was clicked and takes an appropriate action based on which button was clicked and if it matches the player.

<u>Problems:</u>

- All of our problems are within the PlayPanel of the GUI. The game runs properly and can be played correctly according to the rules as validated by direct testing of all of the other methods(PlayGame, Player, Pawn, Board, ComputerPlayer, etc). However, the interface has several problems which prevent the game from displaying properly in the GUI as it is played.
- Sometimes, clicking on the Pawn is not effective and doesn't result in the move being displayed on the GUI even if a move should be allowed according to the rules of the game and takes place within PlayGame
- Turns switch without allowing players to make their proper move
- Players' turns don't update properly using the 'turn' label, resulting in the GUI displaying a label indicating that the same player can take multiple turns in a row
- Blue Pawn disappears after being clicked on to move but does reappear in its proper position on the board(see below)



- Checks which are supposed to prevent the player from starting the game if they have not added their Pawns using the Setup Panel and to prevent the Player from making an illegal move which will result in overlapping Pawns of their own color, do not work properly