Lew, Phoebe

CSCI 103

Crowley, TTH

Design write-up PA6

**System Name:** Board Game

**System Description:**

This system implements a board game where 2-4 players try to reach the end of the board first. All players start out on the same square (SQ01). To advance, they roll two six sided dice. Depending on which square they land on, a variety of different actions can occur. 1) They have to draw a card from the "happy deck". 2) They have to draw a card from the "sad deck". 3) They must either move forward or backward several squares. 4) They pick up or lose stars. 5) Nothing happens, it is a safe space. 6) Roll the dice again. However, if you land on a movement square and you are moved to another movement square, nothing happens. Only one action can occur per turn. The same applies to roll again actions. If you roll again and land on a movement square, for example, you do not take that movement. You just stay where the second dice roll sent you.

The game ends when:

1. All the players except one resigns, aka a player is playing by themselves
2. When a player reaches the last square

The player who reaches the last square the first is the winner of the game. Everyone gets pats on the back for collecting (or not collecting) stars.

//////////////////////////////////////////////////////

That's how the game should work in theory, but I gave up trying to implement it in QT. At some point, the changes I would make in the .cpp files wouldn't be reflected in the QT windows. Program compiles, but it pretty much doesn't do anything. Look through the .cpp/.h files if you want.

//////////////////////////////////////////////////////

**Global Data (in pa6.cpp):**

numPlayers – keeps track of how many players the user wants

numQuit – keeps track of how many players have quit, because if there is only one player remaining, the game should end

Bank bank – creates a bank object to be used later

Char playing – character used to keep track of whether the player wants to quit. If the player inputs a 'q', it means the quit

Action\* boardActions[43] – an array pointers to all the actions. This array corresponds to the each space and each action the space should have.

Spaces[43] – an array of all the spaces. These are what will be printed out in the game board

Player players[4] – an array of all the players (game can be played with maximum 4 players)

happyDeck[4] - the "deck" of cards. This is the first deck, "happydeck".

sadDeck[4] – the second deck of cards, "sadDeck"

**Function/Class Descriptions:**

**Class: Player**

Public:

* Player():
  + Parameters and types: none, this is the default constructor
  + Return type: none, creates an instance of Player
  + Function description: This function initializes the private variables in the Player class. The player starts at space 0, with 0 points, and with a blank space as their default piece.
* selectPiece(string choice):
  + Parameters and types: takes in a string as a parameter
  + Return type: void
  + Function description: After putting in a string as an argument (choice), this function assigns the string as the player's piece.
* showPiece():
  + Parameters and types: none
  + Return type: string
  + Function description: This function is a "getter". It is used in the game\_board class to return the piece corresponding to the player.
* getSpace():
  + Parameters and types: none
  + Return type: int
  + Function description: This function returns the number of the space the player is currently on.
* setSpace(int space):
  + Parameters and types: takes an integer
  + Return type: void
  + Function description: This function is used to update which space the player is moving to. The argument specifies the space number.
* getStars():
  + Parameters and types: none
  + Return type: int
  + Function description: This function returns the total number of points/stars the player has.
* setStars(int num):
  + Parameters and types: takes one integer
  + Return type: void
  + Function description: this function updates how many stars the player has. The total is either incremented or decremented based on the argument "num".

Private variables:

* \_currentSpace: this is an integer that shows what space the player is currently on
* \_totPoints: this is an integer that tracks how many points the player has
* \_piece: currently represented by a string, each player picks a "piece" that will be used to represent them on the game board.

**Class: Space**

Public:

* Space()
  + Parameters and types: none
  + Return type: none, this is the default constructor
  + Function description: This function initializes the private variables of the space. The name is set to 0 and the Boolean array is set to false, meaning that there are no players on the space.
* populate(Action\*\* boardActions, int i, int numPlayers)
  + Parameters and types: This function takes in a double pointer to an array of pointers to the class Action called boardActions, an integer i, and an integer that counts how many players are playing the game.
  + Return type: void
  + Function description: This function fills all 43 spaces with the correct data to be displayed on the gameboard. The action corresponding to the space is linked with the pointer actions. Now, \_myAction points to the corresponding element in the actions array, which was created as a global variable in pa5.cpp. The function then names each square by their number. Finally, for the very first square, it sets the array of booleans to true because all players start out on the first square.
* getPlayer(string p, int posit):
  + Parameters and types: This function takes in a string, p, and an integer, posit.
  + Return type: string
  + Function description: This function is used to return the player's piece to be displayed on the game board. It is called in game\_board.cpp, where a particular player is specified and their piece given as an argument (string p), as well as the position that player corresponds to in the Boolean array in the space class. If a player is on that square, then their piece is returned. If not, the function returns a blank space.
* getName():
  + Parameters and types: none
  + Return type: string
  + Function description: This function returns the name of a space. It is used to print the board, which consists of all the squares and their names.
* getAction(int j):
  + Parameters and types: This function takes in an integer j
  + Return types: string
  + Function description: This function returns the action text corresponding to a particular space. It uses the \_myAction pointer to call getText(j), which exists in the action class. The j is used to specify which line of text to return (there are 3 lines of text for every action).
* ifOccupied(int i):
  + Parameters and types: Takes in an integer
  + Return types: bool
  + Function description: In main, this function is used to determine whether the game is over or not. Namely, it returns true if a player is currently occupying a space and false if he/she is not. The ending condition for the game is if any player is occupying the last square, so this function checks if any of the player positions for spaces[42] is true.
* Toggle(bool switchTo, int player):
  + Parameters and types: Takes a bool and an integer specifying which player
  + Return type: void
  + Function description: The function switches or "toggles" whether a particular player's spot on a given space is occupied or not. switchTo specifies whether a player should be removed from the space (set to false), or put on the space (set to true). The player int determines which player's Boolean should be set to true/false.

Private variables:

* \_name: this is a string that stores the name of the space
* \*\_myAction: this is a pointer to the array of actions created in pa5.cpp (global variable)
* \_players[3]: this is an array of booleans used to keep track of whether or not a player is on a particular space. Each element in the array corresponds to a particular player. (First element is for player 1, second is for player 2, etc.) By default, all the spaces except the first are set to false, meaning that there are no players on the space. The first is set to true because all players start out on the first space.

**Class: Action**

Public:

* Action():
  + Parameters and types: none
  + Return type: none, this is the default constructor
  + Function description: This function creates an instance of the class Action. For this PA, no actions are displayed, so the all elements in the \_text array are filled with blanks.
* setBoardAction(int i):
  + Parameters and types: This function takes in an integer i, used to determine which square's action is being set
  + Return type: void
  + Function description: This function sets the text for each of the squares to be printed out in the printBoard functions. The argument integer is used to determine which square (and therefore what text) will be printed.
* getText(int j):
  + Parameters and types: This function takes in an integer j (it is the same integer as in getAction() in the above class).
  + Return types: string
  + Function description: This function returns one line of text (a string) corresponding to that action. Which line of text is specified by j.

Private:

* \_text[3]: This is an array of strings used to store the text describing the action corresponding to a particular space.

**Class: Game\_Board**

Public:

* Game\_Board(space\* spaces):
  + Parameters and Types: This function takes in a pointer to an array of space objects.
  + Return type: none, this is the default constructor
  + Function description: The only thing this function does is make the \_spaces pointer point to the array of spaces created as a global variable in pa4.cpp
* printFirst(string p1Piece, string p2Piece, string p3Piece, string p4Piece):
  + Parameters and Types: This function takes in four strings. Each string represents a player's piece.
  + Return type: void
  + Function description: This function prints the first line of the board. The board is divided into three sections or "patterns" that can be printed in the same way; this is the first section.
* printMiddle(bool last, string p1Piece, string p2Piece, string p3Piece, string p4Piece):
  + Parameters and Types: This function takes in the same four strings as the printFirst() function. In addition, it takes in a Boolean variable "last", that determines whether or not it is the second to last line being printed.
  + Return type: void
  + Function description: This function prints the middle five lines of the board. If the Boolean is false, it does not print some underscores due to the pattern of the board. If the Boolean is true, it prints an additional line to complete the bottom of the board.
* printLast(string p1Piece, string p2Piece, string p3Piece, string p4Piece)
  + Parameters and Types: This function takes in the same four strings as the above two functions.
  + Return type: void
  + Function description: this function prints the last line of the board.
* printBoard(string p1Piece, string p2Piece, string p3Piece, string p4Piece)
  + Parameters and Types: This function takes in the same four strings as above. This function is actually the one passing the strings to the above three functions.
  + Return type: void
  + Function description: This function prints the board from left to right, top to bottom. It calls on printFirst(), then printMiddle(), with it *not* being the last "middle" line, then printMiddle(), with it being the last "middle" line, and finally, printLast(), which prints the last line. This is the only function in the game\_board class directly called in pa5.cpp

Private:

* Space\* \_spaces: This variable is a pointer to an array of spaces

**Class: Bank**

Public:

* Bank():
  + Parameters and types: none
  + Return type: none, this is the default constructor
  + Function description: The number of stars the bank starts out with is 100000000. It should not run out over the course of the game.
* executeAction(int space, int stars, int i, Player\* player, Space\* spaces)
  + Parameters and types: of the included parameters, the ones used in the function are int space, Player\* player (pointer to array of Player type), and Space\* spaces (pointer to array of Space type), and int i.
  + Return types: void
  + Function description: This is part of the 4 actions our gameplay must have. It is for squares that give/take stars from players. In either case, the function uses the setStars() function in the player class to adjust the number of spaces. At the end, it updates which space the player should appear on because this happens regardless of whether or not a player received stars or had them taken away.

Private:

* Int \_Stars: This variable is the number of stars the bank contains.

**Class: MoveAction**

Public:

* MoveAction():
  + Parameters and types: none
  + Return type: none, this is the default constructor
  + Function description: none
* executeAction((int space, int stars, int i, Player\* player, Space\* spaces):
  + Parameters and types: int space specifies which space number should be used in the function, int stars is the number of stars to be dealt with, int i us used to specify the player number, Player\* player is a pointer to the array of players, Space\* spaces is a pointer to the array of spaces
  + Return type: void
  + Function description: This is one of the 4 actions to be involved in gameplay. If you land on a space that requires you to move to another space, this executeAction updates the position of the player in the player class itself, and where the player piece will be displayed on the gameboard (by changing the Boolean in the spaces array).

Private: none

**Class: card1**

Public:

* Card1():
  + Parameters and types: none
  + Return type: none, this is the default constructor
  + Function description: This just sets the card text as a bunch of spaces (blank).
* setCards():
  + Parameters and types: none
  + Return type: void
  + Function description: This sets the card text to be losing 20 stars.
* getText():
  + Parameters and types: none
  + Return type: string
  + Function description: This function returns the text of the card to be displayed to the player when they draw the card.
* executeAction((int space, int stars, int i, Player\* player, Space\* spaces):
  + Parameters and types: The only variables used in the function itself are int i, the pointer to an array of players (player), and the int giving the number of stars.
  + Return type: void
  + Function description: Using i to specify which player should have their number of stars adjusted, this function updates the player's total number of stars using the setStars() function in the player class.

Private:

* String \_text: this variable displays the action that the card tells the player to take.

**Class: card2**

Public:

* Card2():
  + Parameters and types: none
  + Return type: none, this is the default constructor
  + Function description: This just sets the card text as a bunch of spaces (blank).
* setCards():
  + Parameters and types: none
  + Return type: void
  + Function description: This sets the card text to be adding 17 stars.
* getText():
  + Parameters and types: none
  + Return type: string
  + Function description: This function returns the text of the card to be displayed to the player when they draw the card.
* executeAction((int space, int stars, int i, Player\* player, Space\* spaces):
  + Parameters and types: The only variables used in the function itself are int i, the pointer to an array of players (player), and the int giving the number of stars.
  + Return type: void
  + Function description: Using i to specify which player should have their number of stars adjusted, this function updates the player's total number of stars using the setStars() function in the player class.

Private:

* String \_text: this variable displays the action that the card tells the player to take.

**Function: main() (in pa5.cpp)**

* Parameters and types: none
* Return type: int
* Function description: This is the function where the game is actually played. First, it sets up all of the necessary components of the game. It fills in all the spaces on the board with their corresponding action types. It also generates the two decks of cards to be used in gameplay. After prompting the user, the program stores the number of players specified by the user as numPlayers. Then, it lets each player select their game piece. No two players may have the same piece. When all the players have selected their piece, the previously empty "spaces" array is populated and the gameboard is printed. Each player's piece is also stored in strings to make printing the board easier.

We are now ready to play the game. The game should be played until a player reaches the last square, which is the condition in the while loop encompassing all of the gameplay code. Each player can choose whether or not they want to quit at the beginning of their turn. If they do, the remaining players are printed out and the turn moves to the next person. The piece corresponding to the player who quitted will also no longer appear on the gameboard. If there is only one player remaining after another player quits, the game ends. Since no one reached the end of the board, there is no winner.

Otherwise, the gameplay continues. The player is first removed from their original space. Depending on which square they land on, the appropriate executeAction is called. When that player's turn ends and their actions are executed, it is the next player's turn. This continues until a player reaches the end of the gameboard. After exiting the while loop, the winning player and all the players' star totals are printed out.

**Function: dice() (in pa5.cpp)**

* Parameters and types: none
* Return type: int
* Function description: This function simulates rolling dice. It generates two random numbers between 1 and 6 (inclusive) and adds them together. Then, it returns this value.

**Compiling and Testing**

Compile as follows:

$qmake -project

$qmake

$make

$./pa6