Lew, Phoebe

CSCI 103

Crowley, TTH

Design write-up PA4

**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. In this PA, a points system and actions for each square have not been added.

**Global Data (in pa4.cpp):**

numPlayers – keeps track of how many players the user wants

move – will be used to track how many squares a player should move after rolling the dice

p1 – player 1

p2 – player 2

p3 – player 3

p4 – player 4

actions[43] – an array of all the actions. This array corresponds to the each space.

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

**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.

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\* actions, int I, int numPlayers)
  + Parameters and types: This function takes in a pointer to an array of the class Action called actions, 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 pa4.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).

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 pa4.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.
* 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 pa4.cpp

Private:

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

**Function: main (in pa4.cpp)**

* Parameters and types: none
* Return type: int
* Function description: This is the function where the game will actually be played. First, it stores the number of players as numPlayers. Then, it lets each player select their game piece. No two players may have the same piece. Then, it fills in all the spaces and prints the game board. The players begin on the first square.

**Compiling and Testing**

A makefile is included, but all .o files are already in the folder.

Just run the program with ./pa4

When prompted, enter the number of players between 2 and 4.

The program will then ask the first player what piece they would like to use from a set of 10 different game pieces. After choosing, the second player will be prompted to choose a piece. After all players have chosen a different piece, the game board will be displayed with all the players on the first/starting square.

\*\*\*NOTE\*\*\*

If the game board does not display properly/is cut off, zoom out in the terminal. It should look like this:

