John Pugliesi
Crowley
MW 12-2
PA5

Simponopoly

System Description

Simponopoly is an extremely simplified version of the board game “Monopoly.” In its current state, the game consists of 2-4 players who can move around the board endlessly. The game exits when the user indicates that they do not wish to continue.

This document has information on the functions, data, and classes used throughout the game. Each header in this design writeup corresponds to a .cpp file/class file used in the program.

Function/Class Descriptions

simponopoly.cpp

Global variables

const int NUM_PIECES = 10	constant for number of pieces
std::string pieces[NUM_PIECES]	array of pieces that players can use
int numPlayers	number of players in game start
std::vector<Player> players;	array of those players
int turn = 0;	turn of play
Bank theBank;	The bank object
Deck theMan;	Deck of “*THE MAN*” Cards
Deck chest;	Deck of “*Chest*” Cards

Function Name: main

- **Formal Parameters:** none
- **Return Type:** int
- **Description:** occupies the array of pieces, gets the number of players that are playing in the game, has them choose their pieces, sets up the game board, and contains the loop for the gameplay.

Function Name: welcome()

- **Formal Parameters:** none
- **Return Type:** int
- **Description:** prints a welcome message, gets user’s option to play the game

Function Name: gameOver()

- **Formal Parameters:** none
- **Return Type:** int
- **Description:** checks if game is over, or if user quits. Game is over if only one player remains. returns 1 for game over, 0 otherwise.

Function Name: populateDecks()

- **Formal Parameters:** Game_Board* theBoard: board pointer
- **Return Type:** void
- **Description:** adds and shuffles cardActions to the respective "*THE MAN*" and "Chest" decks in the Game_Board class.

Function Name: getTurnOption()

- **Formal Parameters:** none
- **Return Type:** int
- **Description:** Retrieves reply from each player, determining whether or not they want to continue play when their respective turn is up

Function Name: playTurn()

- **Formal Parameters:**
  GameBoard* theBoard: pointer to the board
  Player* thePlayer: pointer to the player with the turn
  Die d1: first die
  Die d2: second die
  int turn: number turn the game is on
- **Return Type:** int
- **Description:** Executes a turn, moving the player after a dice roll, presenting the player with different options depending on the space the land (buy property, sell property, upgrade property)

Action.cpp

protected members:

string name : the name of the action
string description: descriptor of action
int value = holds monetary value of action, if there is any
Player* actingPlayer : player who will be acted upon
Player* recievingPlayer : potential second player to be included in the action

public members:

Function Name: Action()

- **Formal Parameters:** none
- **Return Type:** none
- **Description:** constructs empty Action object

Function Name: Action(string name, string category)

- **Formal Parameters:** none
- **Return Type:** none

- **Description:** Constructor that makes action of specific name and category

Function Name: ~Action()
- **Formal Parameters:** none
- **Return Type:** none
- **Description:** Deconstructor, does nothing as of now

Bank.cpp

private members:

int amount : holds amount of money bank has

public members:

Function Name: Bank()
- **Formal Parameters:** none
- **Return Type:** none
- **Description:** constructs Bank object with amount of 100000

Function Name: ~Bank()
- **Formal Parameters:** none
- **Return Type:** none
- **Description:** Deconstructor, does nothing as of now

Function Name: withdraw()
- **Formal Parameters:** int withdraw
- **Return Type:** int
- **Description:** returns amount requested to withdraw, and decreases bank amount by that size

Function Name: deposit()
- **Formal Parameters:** int deposit
- **Return Type:** int
- **Description:** returns amount deposited, adds that amount to bank's amount

Function Name: getName()
- **Formal Parameters:** none
- **Return Type:** string
- **Description:** retrieves action name

Function Name: virtual executeAction()
- **Formal Parameters:** none
- **Return Type:** void
- **Description:** abstract method for executing an action

Function Name: virtual executeAction(Player*)
- **Formal Parameters:** none
- **Return Type:** void
- **Description:** abstract method for executing an action while specifying actingPlayer

Function Name: setValue() and GetValue()
- **Formal Parameters:** int, none
- **Return Type:** void, bool
- **Description:** getter and setter for the value of an action

Function Name: setActingPlayer, setRecievingPlayer()
- **Formal Parameters:** Player* newPlayer: player pointer
- **Return Type:** void
- **Description:** setters for acting and receiving players

Function Name: getActingPlayer, getRecievingPlayer()
- **Formal Parameters:** none
- **Return Type:** Player*
- **Description:** getter for acting and receiving players

Card.cpp

private members:

string name: the name

Action* action: the action of the card

public members:

Function Name: Card(string, Action*)
- **Formal Parameters:** string name, Action* action
- **Return Type:** none
- **Description:** constructs card with name and action

Function Name: getAction, setAction()
- **Formal Parameters:** none, Action*
- **Return Type:** Action*, void
- **Description:** setter and getter for a card's action

Function Name: getName()
- **Formal Parameters:** none
- **Return Type:** void
- **Description:** getter for a card's name

CardAction.cpp

private members:

Deck* deck : pointer to the deck in which the card action resides

public members:

Function Name: Card(Deck* theDeck)
- **Formal Parameters:** Deck*
- **Return Type:** none
- **Description:** constructs cardAction in specific deck

Function Name: executeAction()
- **Formal Parameters:** none
- **Return Type:** void
- **Description:** abstract method for executing the cardAction

Function Name: executeAction(Player*)

- **Formal Parameters:** none
- **Return Type:** void
- **Description:** method for executing an action while specifying actingPlayer for the cardAction

Die.cpp

private members:

int numSides : number of sides on the die

public members:

Function Name: Die**()**

- **Formal Parameters:** none
- **Return Type:** none
- **Description:** constructs die with numSides = 6

Function Name: ~Die()

- **Formal Parameters:** none
- **Return Type:** none
- **Description:** Deconstructor, does nothing

Function Name: rollDie()

- **Formal Parameters:** none
- **Return Type:** int
- **Description:** generates random number between 1 and numSides, and returns it

Deck.cpp

private members:

vector<Action*> cards;

public members:

Function Name: Deck**()**

- **Formal Parameters:** none
- **Return Type:** none
- **Description:** constructs an empty deck vector

Function Name: shuffle()

- **Formal Parameters:** none
- **Return Type:** void
- **Description:** shuffles the deck

Function Name: getAndExecuteCard(Player* p)

- **Formal Parameters:** none
- **Return Type:** void
- **Description:** calls a card's executeAction() function on a given player p

Function Name: addCard(Action* newAction)

- **Formal Parameters:** Action* newAction : sets a new CardAction in the deck
- **Return Type:** void

- **Description:** adds a new CardAction to the deck

Game_Board.cpp

private members:

const static int SPACE_WIDTH = 12 : number of characters wide a space is
const static int SPACES_IN_ROW : number of spaces in a row on the board
int num_players : number of players in the game
Space spaces[NUM_SPACES] : array of spaces for the board

Bank* theBank : pointer to the main bank object
Deck* theManDeck : pointer to the Deck of “THE MAN” cards
Deck* theChestDeck : pointer to the deck of “Chest” cards

Function Name: occupySpaces()
- **Formal Parameters:** none
- **Return Type:** void
- **Description:** fills spaces array with predefined spaces

Function Name: centerString
- **Formal Parameters:** string stringToCenter - the string to center
- **Return Type:** string
- **Description:** returns string centered between board width

public members:

Function Name: Game_Board
- **Formal Parameters:** none
- **Return Type:** none
- **Description:** default constructs gameboard with nothing

Function Name: Game_Board
- **Formal Parameters:** int numPlayers - number of players to start
  Bank* theBank pointer
  Deck* theMan pointer to THE MAN deck
  Deck* theChest pointer to the Chest deck
- **Return Type:** none
- **Description:** initializes num_players to the numPlayers and occupies spaces of the spaces array, while setting bank and deck pointers for reference

Function Name: printBoard()
- **Formal Parameters:** none
- **Return Type:** void
- **Description:** prints the board

const static int NUM_SPACES = 40 : constant of 40 spaces on the board

Function Name: printBoardRowReverse
- **Formal Parameters:** int startIndex - index from which to start printing, int endIndex, bool reverse - whether to print those indexes least to greatest or greatest to least if reverse is true

- **Return Type:** void
- **Description:** prints out spaces to standard output

Function Name: printMiddleRow
- **Formal Parameters:** int startIndex - index from which to start printing, int endIndex
- **Return Type:** void
- **Description:** prints out spaces that occupy the middle rows

Function Name: findSpaceByIndex
- **Formal Parameters:** int index - space to retrieve's index
- **Return Type:** Space*
- **Description:** returns pointer to Space object with corresponding index

GoToAction.cpp

private members:

Space* currentSpace; space player is currently on pointer
Space* newSpace; : space to move to pointer
Game_Board* theBoard : board pointer
Bank* theBank : bank pointer

public members:

Function Name: GoToAction(Player*, Game_Board*, Space*, (std::string desc)
- **Formal Parameters:** Player* : player to move
  Game_Board: board
  Space: Space to move to
  (Optional) desc : description of GoTo event
- **Return Type:** none
- **Description:** constructs cardAction in specific deck

Function Name: executeAction()
- **Formal Parameters:** none
- **Return Type:** void
- **Description:** executes GoToAction and moves player to new space

Function Name: executeAction(Player*)
- **Formal Parameters:** Player* p : player to move
- **Return Type:** void
- **Description:** executes GoToAction and moves specified player p to new space

MoneyAction.cpp

private members:

int amount;
bool pay: paying player if true, false otherwise

public members:

Function Name: MoneyAction(Player* p, int amount, bool paying, (std::string desc)

- **Formal Parameters:** Player* : player to pay/take money from
  Game_Board: board
  Space: Space to move to
  (Optional) desc : description of GoTo event
- **Return Type:** none
- **Description:** constructs moneyAction whether to take or pay money to a player

Function Name: executeAction()

- **Formal Parameters:** none
- **Return Type:** void
- **Description:** pays or takes money from a player depending on bool value of paying

Function Name: executeAction(Player*)

- **Formal Parameters:** Player* p : player to move
- **Return Type:** void
- **Description:** pays or takes money from the specified player p depending on bool value of paying

MoveAction.cpp

private members:

int amount : num spaces to move
Game_Board* theBoard : pointer to the board
Bank* theBank : theBank pointer

public members:

Function Name: MoveAction(Player* p, Game_Board*, int moveValue, (std::string desc)

- **Formal Parameters:** Player* : player to move
  Game_Board: board
  int moveVlaue: num spaces to move
  (Optional) desc : description of GoTo event
- **Return Type:** none
- **Description:** constructs action to move player moveValue number of spaces

Function Name: executeAction()

- **Formal Parameters:** none
- **Return Type:** void
- **Description:** executes Move and moves player to new space

Function Name: executeAction(Player*)

- **Formal Parameters:** Player* p : player to move
- **Return Type:** void
- **Description:** executes MoveAction and moves specified player p to new space

Player.cpp

private members:

string piece : player's chosen piece
int money : amount of money a player has
int currentPosition : current index position of player on the board

public members:

Function Name: Player
- **Formal Parameters:** none
- **Return Type:** none
- **Description:** default constructor initializes Player with empty string for piece and 5000 for money

Function Name: ~Player
- **Formal Parameters:** none
- **Return Type:** none
- **Description:** deconstructor for player, does nothing

Function Name: getCurrentSpace
- **Formal Parameters:** none
- **Return Type:** int
- **Description:** returns player's current space index position

Function Name: move
- **Formal Parameters:** int num - spaces to advance player's index position
- **Return Type:** int
- **Description:** moves player by value num, incrementing their currentPosition value

Function Name: setPosition
- **Formal Parameters:** int index - index position to set player at
- **Return Type:** int
- **Description:** sets currentPosition to specifically provided index

Function Name: setPiece
- **Formal Parameters:** string newPiece - new Piece to set as piece
- **Return Type:** void
- **Description:** sets player's piece

Function Name: getPiece
- **Formal Parameters:** string piece
- **Return Type:** string
- **Description:** returns player's piece

Function Name: giveMoney
- **Formal Parameters:** int amount
- **Return Type:** int
- **Description:** gives the player money of amount 'amount' and returns new money total

Function Name: takeMoney
- **Formal Parameters:** int amount
- **Return Type:** int
- **Description:** subtracts player's money by amount and returns the new money total

PropertyAction.cpp

private members:

Space* currentSpace : current space to transact upon
bool willBuy : player is buying the space if true, selling if not
bool isFromBank : if true, selling/buying to the bank
bool isTransfer : if true, transfer all player's property to bank or other player
Bank* theBank : pointer to the bank
int value : value of the transfer

public members:

Function Name: PropertyAction(Player* p1, Player* p2, Space* cs, Bank* bank, bool buyingIt, bool isbank, bool transfer, (std::string desc)

- **Formal Parameters:** Player* p1 : player to buy/sell/transfer from/to
  Player* p2 : receiving player of transaction
  Space* cs : current space to transact upon
  Bank* : the Bank
  bool buyingIt : see above (private members)
  bool isBank : see above
  bool isTransfer : see above
- **Return Type:** none
- **Description:** constructs Property Action with appropriate parameters to execute a property transaction

Function Name: executeAction()

- **Formal Parameters:** none
- **Return Type:** void
- **Description:** executes PropertyAction and sells/buys/transfers properties to and from provided players depending upon the provided parameters

Function Name: executeAction(Player*)

- **Formal Parameters:** Player* p : player to move
- **Return Type:** void
- **Description:** executes PropertyAction and sells/buys/transfers properties to and from player* p depending upon the provided parameters

Space.cpp

private members:

string name : name of the space
string upgrade: string of upgrade value
int value : value of space
int rent : space's rent
int index : index of space on the board
Action* action : space's action
int nextSpace : space index that comes after

bool ownable : whether or not the space is ownable
Player currentPlayers[4] : array of Players currently on the space
Player* owner : player who owns the space
int numPlayersOnSpace : number of players on the space

public members:

Function Name: Space
- **Formal Parameters:** none
- **Return Type:** none
- **Description:** creates blank ownable space

Function Name: Space
- **Formal Parameters:** string newName - name of space, int nextSpace - space index that comes after - bool canOwn - whether or not space is ownable
- **Return Type:** none
- **Description:** creates space with name newName, and sets whether it can be owned or not

Function Name: isOwnable
- **Formal Parameters:** none
- **Return Type:** bool
- **Description:** returns whether or not space is ownable

Function Name: addPlayerToSpace
- **Formal Parameters:** Player aPlayer - player to add to space
- **Return Type:** void
- **Description:** adds the player to the space

Function Name: getPlayersOnSpace
- **Formal Parameters:** Player aPlayer - player to add to space
- **Return Type:** void
- **Description:** returns array of players on the space

Function Name: playersOnSpaceToString
- **Formal Parameters:** none
- **Return Type:** string
- **Description:** returns string of players currently on the space

Function Name: removePlayerFromSpace
- **Formal Parameters:** Player toRemove - player to remove
- **Return Type:** void
- **Description:** removes player from currentPlayers array

Function Name: upgradeSpace()
- **Formal Parameters:** none
- **Return Type:** void
- **Description:** upgrades space for an owner

Function Name: getUpgradeValue()
- **Formal Parameters:** none
- **Return Type:** int
- **Description:** gets cost of upgrading the space

Function Name: executeAction()

- **Formal Parameters:** none
- **Return Type:** void
- **Description:** calls the space's action's executeAction() method

Function Name: getRent()
- **Formal Parameters:** none
- **Return Type:** int
- **Description:** gets cost of renting the space

Function Name: getValue()
- **Formal Parameters:** none
- **Return Type:** int
- **Description:** gets value of the space

Function Name: getSpaceIndex()
- **Formal Parameters:** none
- **Return Type:** int
- **Description:** gets index of space on the board

Compiling and Running the Game

Compiling and running the game is simple. Follow these steps:

1. Navigate to the directory in which the simponopoly.tar.gz file is located.
2. In your favorite command line application, execute the following command:
   `tar -xvzf simponopoly.tar.gz`
3. `cd` into the new simponopoly directory
4. On the command line, execute the simple make command:
   `make`
5. Boom, you're done compiling. To run the program, type the following into the command line:
   `./simponopoly`
6. Prepare yourself for the exciting game ahead. The fun is literally never-ending (if you want it to be)!

Sample Execution:

```
$ cd {directory_where_simponopoly.tar.gz_is}
$ tar -xvzf simponopoly.tar.gz
$ cd simponopoly/
$ make
$ ./simponopoly

***** Welcome to Simponopoly! *****
_____________________________________
Choose an option to get started:
      (1) Print Instructions and Play
      (2) Start the GAME!!!
      (3) Quit
2
Enter the number of players: 2
(1) !
(2) @
(3) #
(4) $
(5) %
(6) ^
(7) &
(8) *
(9) +
(10) ?

Player 1, choose your piece: 3

Excellent Choice, Player 1 is now the: #
```

```
(1) !
(2) @
X Taken X #
(4) $
(5) %
(6) ^
(7) &
(8) *
(9) +
(10) ?

Player 2, choose your piece: 4

Excellent Choice, Player 2 is now the: $


(board would print here)

The # is up!
Die rolled: 4 and 5.
The # advances 9 spaces!

(board would print here)

$ is up next...
Ready to continue?
```

The game has an instructions section, select it to view the general rules of the game!