

Plan of Attack

This document outlines our proposed implementation for building the CS 246 final project. We will implement "BuildingBuyer7000".

The structure of this outline is as follows:

1. A main function basic outline which describes the sequence of events that occur during gameplay.
2. A description of critical methods to ensure functionality of specified classes.
3. A Task Delegation List
4. A core requirements flowchar - detailing how we will implement the core requirements of the project - as specified in bb7k.pdf.

Main Function Basic Outline

Program is called from command line with optional arguments (-load file or -test).

- I. -load: file the game is played from the state specified in file.
- II.-test: the values of each die roll can be specified by the player.

Read number of players, player names, and chars(board pieces) from cin.

Array of player objects is created and stored in a field of the board. (This is detailed in flow chart)

The commands are as follows:

roll - theBoard->movePlayer is called. Method first checks if the player is in the Timeline. Method changes the players location pointer and calls the action method on the location pointer with it's self as an argument. The method then calls textDisplay->notify(the appropriate coordinates and char). The board is then printed.

trade <name> <give> <receive> - active player offers to trade <give> with <name> in return for <receive>.
theboard->trade(currentPlayer) called.

improve <property> buy/sell - player->improve() improve method reads in property name and checks if player owns. Checks if player can afford improvement if yes calls
property->improve(buy/sell) this method bill/compensates the owner the appropriate amount. Board->print() is then called from main.

mortgage <property> - player->mortgage() called, reads in name of property, checks if it owns property. If player owns property property->mortgage() is called this method makes the property stop charging tuition and compensates the owner.

unmortgage <property> - player->unmortgage() called, checks if it owns property bills the owner the lent money plus ten percent. Makes the property start charging tuition again.

bankrupt - player->bankrupt() is called. Checks if player owes more money than has. If yes all assets are given to play who is owed. Players must immediately pay 10% of the principle when inheriting a mortgaged property. They may then choose to unmortgage be paying the principle if not they will have to pay the usual principle plus 10% when unmortgaging later.

Assets - calls player->displayAssets(). Does not print if player is deciding how to pay tuition

save - saves the state of the game

next - if player is not in tims line and has rolled this turn continues.

Description of Critical Methods

Board::Board() - NHDeckconfig.txt, SLCDckConfig.txt, boardConfig.txt

- Before every game a new board must be made.
- The board constructor, Board::Board(), is entirely responsible for the new board.
- Board configuration is stored in textfiles. Includes property names, prices, improvement costs, tuition.
- Board::Board() constructs the squares using FileReads objects created using the configuration files.
- Construct DeckBuilder object. Run buildDeck(fileName) where fileName is the name of deck configuration file.

Board::startGame() - cin

- Reads in player names and avatars from cin to construct Player objects and puti them in the players. array.
- randomly decides who goes first. Sets currentActivePlay field.

Board::load() - saveFile.txt

- Creates ReadFile object with saveFill string and uses it to create and new players array.
- Resulting players have avatar, money, position, rollup the rim cups and turns left in the DC tims line as specified in saveFile.
- Order of players in array is same as oder they where read in.
- Game continues with first read in player going first.

Board::roll() - cin

- calls Board::movePlayer(numSpaces) where numSpaces is the sum of two random numbers between 1 and 6.
- runs currentActivePlayer->location->action(). Action() alters the currentActivePlayer appropriately.

Board::movePlayer(int numSpaces)

- moves the currentActivePlayer a numSpaces squares forward.

Board::trade(Player *otherPlayer, String offer, String recieve) - cin
- reads accept/reject from cin.
- if accept takes offer from currentActivePlayer and to otherPlayer the takes
recieve from otherPlayer and gives it to currentActivePlayer.
- if reject do nothing.

Player::payDebt()
- defines local int payment as min(debt, savings). Calls creditor-
>recieveTuition(payment) and decrease savings and debt by payment.
- if all debt is paid creditor is assigned to null

Player::recieveTuition(int tuition)
- increases savings by tuition.

Bankrupt()
- All assets are given to the creditor using the trade member of the board.

List of Responsibilities

Specifies what classes each partner is responsible for implementing.

Amanda:

Textdisplay, fileReader, board, player, save files, card config files, property
config files.

Travis:

mainfunction, card and all subclasses, deckBuilder, Square and all subclasses,
TimCupDispenser.