CS246 Spring 2020 Project – Watopoly

C. Kierstead G. Richards G. Tondello

Due Date 1: Wednesday, August 5, 5:00pm
Due Date 2: Saturday, August 15, 11:59pm

DO NOT EVER SUBMIT TO MARMOSET WITHOUT COMPILING AND TESTING FIRST. If your final submission doesn't compile, or otherwise doesn't work, you will have nothing to show during your demo. Resist the temptation to make last-minute changes. They probably aren't worth it.

This project is intended to be doable by three people in two weeks. Because the breadth of students' abilities in this course is quite wide, exactly what constitutes two weeks' worth of work for three students is difficult to nail down. Some groups will finish quickly; others won't finish at all. We will attempt to grade this assignment in a way that addresses both ends of the spectrum. You should be able to pass the assignment with only a modest portion of your program working. Then, if you want a higher mark, it will take more than a proportionally higher effort to achieve, the higher you go. A perfect score will require a complete implementation. If you finish the entire program early, you can add extra features for a few extra marks.

Above all, MAKE SURE YOUR SUBMITTED PROGRAM RUNS. The markers do not have time to examine source code and give partial correctness marks for non-working programs. So, no matter what, even if your program doesn't work properly, make sure it at least does something.

The Game of Watopoly

In this project, you will implement the video game Watopoly, which is a variant of the game, Monopoly, with a board based on the University of Waterloo campus. If you are unfamiliar with Monopoly, please see `http://en.wikipedia.org/wiki/Monopoly` for an overview. It would be helpful to play a game or two of Monopoly to become familiar with gameplay. You should also read the official Monopoly rule book.

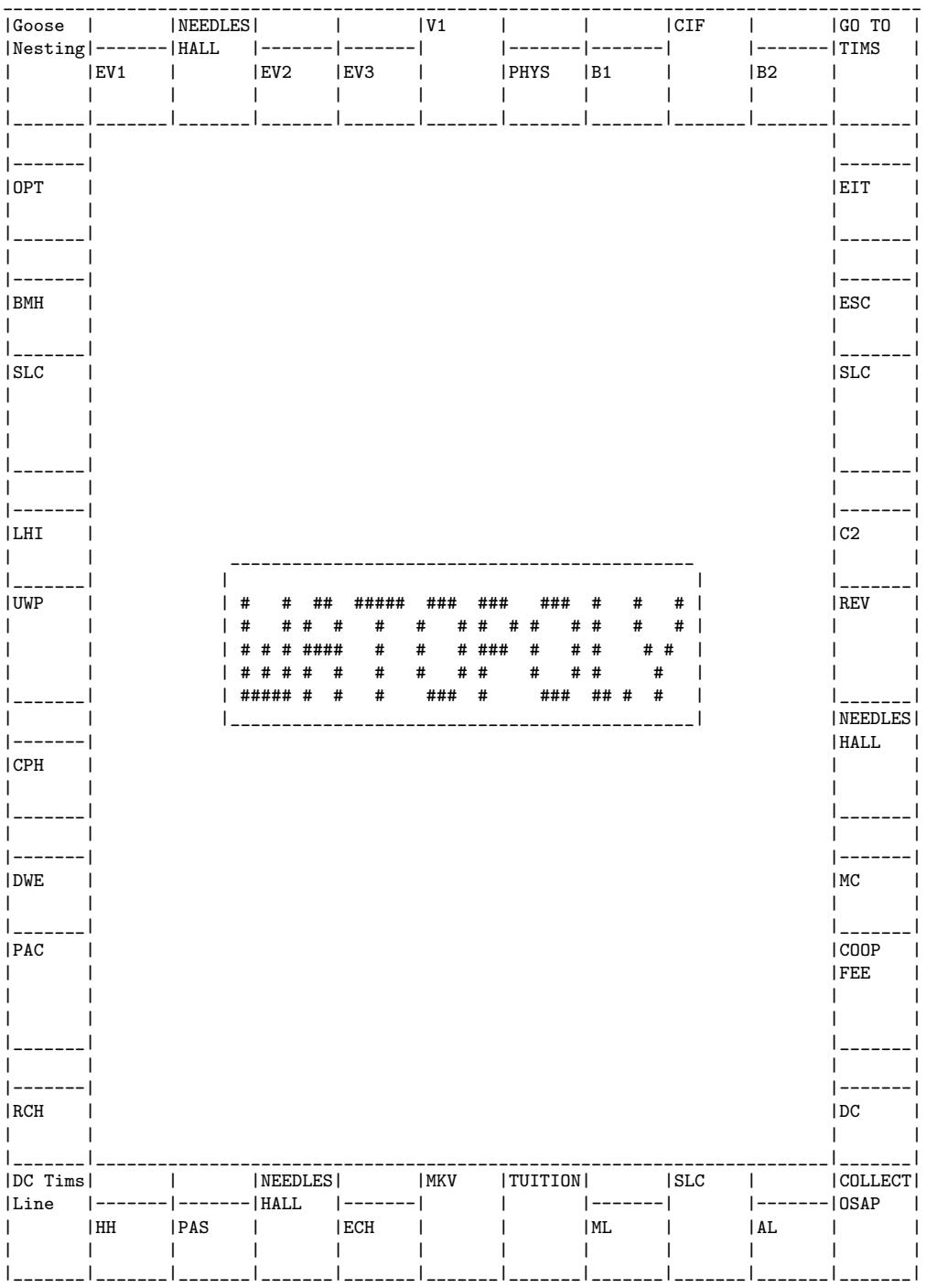
A game of Watopoly consists of a board that has 40 squares. When a player moves, an action specific to the square they land on occurs. The goal of the game is to be the only player to not drop out of university (declare bankruptcy). Play proceeds as follows: players take turns moving around the board, buying and improving on-campus buildings (properties), and paying tuition (rent).

System Components

The major components of the game are as follows:

Board

The board is the visual representation of the state of the current game. The board should always display the 40 squares, any improvements for an academic building, and a representation for each player in the game which shows their current location on the board. There are two types of squares on the board: ownable properties (academic buildings, gyms, and residences) and unownable buildings (e.g. run by administration).


Goose Nesting
NEEDLES HALL
V1
CIF
GO TO TIMS
EV1
EV2
EV3
PHYS
B1
B2
OPT
EIT
BMH
ESC
SLC
SLC
LHI
C2
UWP
REV
NEEDLES HALL
CPH
DWE
MC
PAC
COOP FEE
RCH
DC
DC Tims Line
NEEDLES HALL
MKV
TUITION
SLC
COLLECT OSAP
HH
PAS
ECH
ML
AL

Buildings

Question. After reading this subsection, would the Observer Pattern be a good pattern to use when implementing a gameboard? Why or why not?

The squares on the board represent buildings or locations on campus. Different buildings have different purposes. Academic buildings can be purchased by players, which allows tuition to be charged to other players for attending lectures in said building. Each Academic Building is part of a smaller collection of properties, known as a monopoly. When a player owns multiple buildings from a monopoly, the calculation of tuition changes depending on the type of property[1]. If a player lands on an unowned Academic building, they may choose to purchase the property for its corresponding value. A player may choose not to purchase a property. If this happens the bank will auction the property (see the Auctions section). If the property is owned by another player, the player must pay the appropriate tuition to the owner.

Academic Buildings

The academic buildings on the board are the properties which form monopolies in the following sets: AL and ML, ECH, PAS and HH, RCH, DWE and CPH, LHI, BMH and OPT, EV1, EV2 and EV3, PHYS, B1 and B2, EIT, ESC and C2, and MC and DC. When a player owns all the properties in a monopoly, they can purchase improvements (4 bathrooms, and then a cafeteria) for each building. Tuition is doubled for each building that has no improvements when the monopoly is owned by a single player. The information for each building is as follows:

				Tuition with Improvements					
Name	Monopoly Block	Purchase Cost	Improvement Cost	0	1	2	3	4	5
AL	Arts1	40	50	2	10	30	90	160	250
ML	Arts1	60	50	4	20	60	180	320	450
ECH	Arts2	100	50	6	30	90	270	400	550
PAS	Arts2	100	50	6	30	90	270	400	550
HH	Arts2	120	50	8	40	100	300	450	600
RCH	Eng	140	100	10	50	150	450	625	750
DWE	Eng	140	100	10	50	150	450	625	750
CPH	Eng	160	100	12	60	180	500	700	900
LHI	Health	180	100	14	70	200	550	750	950
BMH	Health	180	100	14	70	200	550	750	950
OPT	Health	200	100	16	80	220	600	800	1000
EV1	Env	220	150	18	90	250	700	875	1050
EV2	Env	220	150	18	90	250	700	875	1050
EV3	Env	240	150	20	100	300	750	925	1100
PHYS	Sci1	260	150	22	110	330	800	975	1150
B1	Sci1	260	150	22	110	330	800	975	1150
B2	Sci1	280	150	24	120	360	850	1025	1200
EIT	Sci2	300	200	26	130	390	900	1100	1275
ESC	Sci2	300	200	26	130	390	900	1100	1275
C2	Sci2	320	200	28	150	450	1000	1200	1400
MC	Math	350	200	35	175	500	1100	1300	1500
DC	Math	400	200	50	200	600	1400	1700	2000

Table 1: Academic Purchase/Tuition/Monopoly Block information

[1]Due to increased maintenance costs from these "improvements"

Residences

The Residences, MKV, UWP, V1, and REV, each cost $200. Rather than paying Tuition visitors will pay Rent to the residence owner. Rent is calculated based on the number of residences a player owns. The rent when one residence is owned is $25, two residences $50, three residences $100, and four residences $200.

Gyms

The two Gyms, PAC and CIF, each cost $150. Usage fees are calculated by rolling two dice. If one Gym is owned by a player, the fee is 4 times the sum of the dice. If two Gyms are owned by a player, the fee is 10 times the sum of the dice.

Non-Property Squares

The following are squares on the board which cannot be bought.

Collect OSAP

Each time a player passes over or lands on the Collect OSAP square, they collect $200 unless told otherwise. Unlike the actual OSAP collection process there are no lines nor lengthy forms to present.

DC Tims Line

When a player lands on the DC Tims Line square, nothing happens. If the player is sent to the DC Tims Line square by any other means, they cannot leave the line unless they roll doubles, pay $50 [2] or use a Roll Up the Rim cup (discussed below). On their third turn of being in the DC Tims Line, if a player does not roll doubles, the player must leave the DC Tims Line, either by paying or using a Roll Up the Rim cup. At this point, they move the sum of the dice from their last roll.

Go to Tims

When a player lands on the Go To Tims square, they are moved to the DC Tims Line square. They do not collect the $200 from Collect OSAP.

Goose Nesting

When a player lands on Goose Nesting, they are attacked by a flock of nesting geese but otherwise nothing happens.

Tuition

Choose between paying $300 tuition or 10% of your total worth (including your savings, printed prices of all buildings you own, and costs of each improvement). All money goes to the School (aka the Bank).

Coop Fee

Immediately pay $150 to the School (aka the Bank).

SLC

When a player lands on an SLC square, their piece is randomly moved on the board. A message should be printed stating by how much the player moves and to which square they are moved. Play proceeds as if they had landed on the square they were moved to. The movements are listed below.

[2] Coffee has gotten really expensive lately.

Move	Probability
Back 3	1/8
Back 2	1/6
Back 1	1/6
Forward 1	1/8
Forward 2	1/6
Forward 3	1/6
Go to DC Tims Line	1/24
Advance to Collect OSAP	1/24

Table 2: SLC actions

Needles Hall

When a player lands on a Needles Hall square, they either gain or lose some amount of their savings. The amount that a player's savings are affected is given by the distribution below.

Change	Probability
−200	1/18
−100	1/9
−50	1/6
25	1/3
50	1/6
100	1/9
200	1/18

Table 3: Needles Hall actions

Question. Suppose that we wanted to model SLC and Needles Hall more closely to Chance and Community Chest cards. Is there a suitable design pattern you could use? How would you use it?

Roll Up the Rim Cup

When landing on either Needles Hall or SLC, there is a rare chance (1%) that instead of the normal effect of visiting that building the player receives a winning Roll Up the Rim cup. This cup is used to get out of the DC Tims line for free. At any point in time there should be no more than 4 cups active (i.e., the total number of cups owned by all players cannot exceed 4, which means that if 4 cups are active there is a 0% chance of receiving a Roll Up the Rim cup).

Players

All players in the game will be "human" players and controlled by the commands described in the Command Interpreter section and any other commands you implement and document. Your game must support at least 6 players. Note that the given display (which you do not need to follow precisely) cannot support the full 8 players that Monopoly supports because of the number of tiles used for each building.

A player chooses from the following pieces to represent them on the board. Only one player may choose a particular piece.

Name	Goose	GRT Bus	Tim Hortons Doughnut	Professor	Student	Money	Laptop	Pink tie
Char	G	B	D	P	S	$	L	T

Command Interpreter

Initially, the game will demand the user enter the number of players, followed by the name of each player and the character that will represent the player on the board. Each player starts with $1500. Play will then continue in the way described below or until one player remains.

The following commands can be supplied to your command interpreter:

- **roll** (used if the player can roll): the player rolls two dice, moves the sum of the two dice and takes action on the square they landed on.
- **next** (used if the player cannot roll): give control to the next player.
- **trade <name> <give> <receive>**: offers a trade to `name` with the current player offering `give` and requesting `receive`, where give and receive are either amounts of money or a property name. Responses are `accept` and `reject`. For example,
  - **trade Brad 500 DC** indicates that the current player is willing to give Brad $500 in exchange for the DC building
  - **trade Rob DC MC** indicates that the current player is willing to give Rob DC in exchange for MC
  - **trade Kevin MC 500** indicates that the current player is willing to give Kevin MC in exchange for $500

  Note: it does not make sense for a player to offer to give money in return for money. Your game should reject such an attempt with an appropriate message and continue.

  Note: A player can only offer to trade a property if all properties in the monopoly (including the property being offered for trade) do not have improvements. In the case that a player incorrectly offers for trade a property that has improvements (or for which a property in the monopoly has improvements), the game must automatically reject the trade offer. A player can offer a trade even when stuck in the DC Tims Line.
- **improve <property> buy/sell**: attempts to buy or sell an improvement for `property`.
- **mortgage <property>**: attempts to mortgage `property`.
- **unmortgage <property>**: attempts to unmortgage `property`.
- **bankrupt**: player declares bankruptcy. This command is only available when a player must pay more money then they currently have.
- **assets**: displays the assets of the current player. Does not work if the player is deciding how to pay Tuition.
- **all**: displays the assets of every player. For verifying the correctness of your transactions. Does not work if a player is deciding how to pay Tuition.
- **save <filename>**: saves the current state of the game (as per the description below) to the given file.

Note that the board should be redrawn as appropriate every time a command is entered. For game play, it may be useful to have other commands which display other information about the game.

It would be in your best interest (and will help during your demo) to make sure that your program does not break down if a command is misspelled.

Gameplay

A typical turn should be played as follows:

When not in the middle of another action, the player can offer a trade, buy/sell improvements, and mortgage/unmortgage buildings they own.

If the player is in the DC Tims Line, they follow the procedure to leave.

Otherwise, the player rolls two six-sided dice. Their token moves the sum of dice. They take the action of the square they landed on. If they rolled doubles, they roll again. If they roll three sets of doubles in a row, they do not move the number shown on the last set and move to DC Tims Line[3] instead and cannot roll again.

When the player cannot roll, they can finish their turn.

At any time when a player must pay more money than they currently have, they have the option to drop out (declare bankruptcy) or attempt to trade, mortgage buildings and sell improvements to gather the required money.

Improvements

A player can build up to 5 improvements on academic buildings when they own a monopoly, (four bathrooms, followed by a cafeteria). The cost of each improvement for an academic building is determined in the table given previously. If a player decides to sell an improvement, they receive half of the cost of the improvement. Each improvement is represented by an `I` in the top row of the property it belongs to in the text display.

Question. Is the Decorator Pattern a good pattern to use when implementing Improvements? Why or why not?

Mortgages

A player can mortgage a building they own at any time. When a property is mortgaged, they receive half of the cost of the property and people who land on the property do not owe rent to the owner. Note that any improvements must be sold (as above) before a building can be mortgaged.

To unmortgage a property, the player must pay half of the cost of the property plus 10% more of the cost of the property (a total of 60% of the cost of the property).

Dropping Out (Bankruptcy)

A player can only drop out (declare bankruptcy) when they owe more money than they currently possess. At this point, the player who owes money has two options: they can either declare bankruptcy or try to raise the money they owe.

If they declare bankruptcy because they owe money to another player, the player who is owed receives all of the assets of the player declaring bankruptcy. Otherwise (if the player owes money to the Bank), the buildings are returned to the open market as unmortgaged properties (see Auctions below) and all Roll Up the Rim cups are destroyed.

Alternately, the player who owes money can first sell off all improvements and mortgage everything in an effort to raise enough money. If they are able to raise enough money, bankruptcy cannot be declared. Otherwise (not enough money could be raised), the player owing the money declares bankruptcy, following the procedure described above.

Note that should a player receive a mortgaged building they must immediately pay 10% (of the original price of the property) to the Bank. They may then choose to unmortgage the property by paying the principal, i.e., half the cost of the property. Should they choose to leave the property mortgaged they must pay an additional 10% (of the original price of the property) should they later decide to unmortgage the property.

[3] They stayed up all night programming and *need* caffeine.

Auctions

If a player decides not to buy a property or a player declares bankruptcy to the bank, properties are auctioned. During an auction, each player is given the option to either raise the current bid or withdraw from the auction. When only one player has not withdrawn from the auction, they are the winner and must pay their final bid.

Ending the Game

The game ends when there is only one player who has not dropped out (declared bankruptcy). This player is the winner.

Command Line Options

Your program **must** have the ability to process two command line arguments.

Loading A Game

The first command line argument specifies a saved game. The command is: **-load file**. This will be used for testing purposes. Since you are allotted a fixed time to show case your project's features, you will most likely lose many marks on the demo if you do not have this ability. You should not deviate from the specified format. The format of this file will be:

```
numPlayers
player1 char TimsCups money position
player2 char TimsCups money position
...
AL owner improvements
ML owner improvements
MKV owner improvements
...
MC owner improvements
DC owner improvements
```

The first line specifies the number of players. This is followed by a line for each player and a line for each ownable building in the order they occur around the board. For a player, char represents the character used to represent the player on the board, money represents their amount of money and position represents which square they are on, starting with Collect OSAP being 0. A player can not start on square 30 (Go to DC Tims Line). If a player is on square 10 (DC Tims Line), their line will look like one of the following:

```
player char TimsCups money 10 0
player char TimsCups money 10 1 num
```

The first line represents that the player's position is the DC Tims Line but they are not actually in the DC Tims Line. The second line represents the player is in the DC Tims line and num is the number of turns they've been there. The value of num must be between 0 and 2, inclusive.

The buildings will be in the order they occur on the board. Each building in the save file is followed by the name of the player who owns the building. If it is unowned, it is followed by `BANK`[4]. `improvements` represents the number of improvements on the property. Improvements can only be between 0 and 5 inclusive (except as outlined below). If the property is not part of a monopoly, improvements must be 0. Residences and Gyms will always be followed by 0. If the building is mortgaged then the improvements field will be -1. The total number of Tims Cups cannot exceed the maximum number of Tims Cups allowed.

Play resumes with the player listed first.

[4]This means you cannot have a player named `BANK`.

Testing Mode

The second command line argument is **-testing**. The only mandatory testing command is an adaptation of roll. This roll will be called `roll <die1> <die2>`. The player will move the sum of `die1` and `die2`, where each is ANY non-negative value and not necessarily between 1 and 6. You may find it useful to implement other commands for testing mode. The normal roll command will be used if the dice are not specified.

Other Options

You may find it useful to implement another optional argument that represents a seed for your random generation (so that you can have reproducible results). This is **not** required.

Grading

Your project will be graded as follows:

Correctness and Completeness	60%	Does it work? Does it implement all of the requirements?
Documentation	20%	Plan of attack; Final design document.
Design	20%	UML; good use of separate compilation, good object-oriented practice; is it well-structured, or is it one giant function?

Even if your program doesn't work, you can still earn marks associated with the Documentation and Design components of the project.

When all else fails...

This is a relatively complex project with many components and large amounts of random generation. If you find yourself running out of time or having trouble here's our suggestion for priorities:

- Loading in a saved game and starting a new game.
- Rolling dice, player movement, and paying rent.
- Buying properties.
- Declaring bankruptcy.
- Saving games.
- SLC and Needles Hall cards.
- Buying improvements.
- Trading.

Accomplishing the first five points should reward you with at least 50% (assuming you have used object-oriented principles to get that far). That is, you will get a higher mark for submitting something that runs but does not implement all the requirements than submitting something that attempts to implement all the requirements but doesn't run.

If Things Go Well

If you complete the entire project, you can earn up to 10% extra credit for implementing extra features. You should provide some way of playing the game with and without enhancements, as this will allow project assessors to accurately determine that you met all of the base requirements. Recompiling is **not** permitted.

The following is a list of possible enhancements that you could develop. However, don't attempt to implement this until you've got the base game working.

- House rules. The option to change some rules before the game starts. Examples:
  - Money collection from Tuition, Coop Fee, and DC Tims Line goes to the center of the board. Landing on Goose Nesting gives a Player all the money in the center.
  - The above except that the center starts with $500. This is replaced if the money is collected.
  - Landing on "Collect OSAP" doubles the amount received
- Implementing Computer players.
- Additional commands for better game play.
- Different board themes.
- Graphic display.
- Implementing SLC and Needles Hall cards similar to Community Chest and Chance cards.
- Implementing the "even build" rule for houses from the game Monopoly, and limiting the number of houses (improvements).

To earn significant credit, enhancements must be algorithmically difficult, or solve an interesting problem in object-oriented design. Trivial enhancements will not earn a significant fraction of the available marks.

Due Dates

Due Date 1: Due on due date 1 is your plan of attack, `plan.pdf`, and initial UML diagram, `uml.pdf`, for your implementation of Watopoly on Marmoset, `CS246_PROJECT`.
Due Date 2: Due on due date 2 is your actual implementation of Watopoly. All `.h`, `.cc` and any text files needed for your project to compile and run should be included in `watopoly.zip`. The zip file must contain a suitable `Makefile` such that typing `make` will compile your code and produce an executable named `watopoly`. In addition, upload `demo.zip`, which contains `demo.pdf` and all necessary files to run your demo (such as saved files, but not the program executable, which will be compiled from the files submitted in `watopoly.zip`), `uml-final.pdf` and `design.pdf` to the appropriate place on Marmoset, `CS246_PROJECT`.

If you have implemented any bonus features, make sure that your submitted demo plan includes a list of all of the described bonus features, and how to run the game to see the effect of the bonus features.

See **project_guidelines-online.pdf** for instructions about what should be included in your plan of attack and final design document.

A Note on Random Generation

To complete this project, you will require the random (or rather, pseudo-random) generation of numbers. You have two options available to you. In `<cstdlib>`, there are commands **rand** and **srand**, which generate a random number and seed the random generator respectively (typically, seeded with **time()** from `<ctime>`). Alternatively, you can use the **prng class** from `prng.h`, which provides an encapsulated random number generating algorithm. Either is fine for the project; it is not required to use one over the other.