Final Project Design Document

Isobelle Wang, Zhige Chen, Lynn Li

Department of Mathematics, University of Waterloo

CS246: Object-Oriented Software Development

April 10, 2023

Introduction

Our team took on the challenge of creating an innovative and engaging final project for our cs246 course by developing the Watopoly game. With meticulous planning, designing, coding, modifying, and updating, we successfully created a unique version of monopoly tailored to the University of Waterloo. We overcame numerous obstacles and added various design patterns to make it user-friendly and entertaining. The game accommodates 2-8 players and is played on a 40 square board with ownable and unownable buildings. The last player standing wins the game, with many extra credit features included. Overall, our Watopoly game is a testament to hard work, determination, and a passion for game design.

Overview

In the entire project, all implementations can be divided into _ major categories:

1. Grid: to initialize a game; to print the display of the game board, player location, and status of each property on the board to the screen; and to save the current grid information and to load an existing game.
2. Player: to make every action including:
    - roll: to roll the dice in every turn if available.
    - move: to move the number of tiles according to the sum of numbers on the dice rolled.
    - trade: to trade with other player by providing money to get a property, or by providing a property to get money.

- improve: to buy/ sell an improvement of a building in a monopoly.
- mortgage: to mortgage a property.
- unmortgage: to unmortgage a property
- bankrupt: to declare bankruptcy when not having enough money to pay a tuition/ residence/ usage fee at another players' owned academic building/ residence/ gym, or to pay a tuition/ coop fee to the school.
- next: to end the player's current round.
- to interact with each property, as described below.

3. Dice: to roll a pair of dice
4. Ownable property: a property which a player can buy/ trade/ improve/ mortgage/ unmortgage.
    - Academic Building:

						Tuition with Improvements					
Tile	Name	Monopoly Block	Puchase Cost	Total Puchase Cost	Improvement Cost	0	1	2	3	4	5
1	AL	Arts1	40		50	2	10	30	90	160	250
3	ML	Arts1	60	100	50	4	20	60	180	320	450
6	ECH	Arts2	100		50	6	30	90	270	400	550
8	PAS	Arts2	100		50	6	30	90	270	400	550
9	HH	Arts2	120	320	50	8	40	100	300	450	600
11	RCH	Eng	140		100	10	50	150	450	625	750
13	DWE	Eng	140		100	10	50	150	450	625	750
14	CPH	Eng	160	440	100	12	60	180	500	700	900
16	LHI	Health	180		100	14	70	200	550	750	950
18	BMH	Health	180		100	14	70	200	550	750	950
19	OPT	Health	200	560	100	16	80	220	600	800	1000
21	EV1	Env	220		150	18	90	250	700	875	1050
23	EV2	Env	220		150	18	90	250	700	875	1050
24	EV3	Env	240	680	150	20	100	300	750	925	1100
26	PHYS	Sci1	260		150	22	110	330	800	975	1150
27	B1	Sci1	260		150	22	110	330	800	975	1150
29	B2	Sci1	280	800	150	24	120	360	850	1025	1200
31	EIT	Sci2	300		200	26	130	390	900	1100	1275
32	ESC	Sci2	300		200	26	130	390	900	1100	1275
34	C2	Sci2	320	920	200	28	150	450	1000	1200	1400
37	MC	Math	350		200	35	175	500	1100	1300	1500
39	DC	Math	400	750	200	50	200	600	1400	1700	2000

    - Residence:

			Tuition with Owned Number			
Tile	Residence	Purchase Cost	1	2	3	4
5	MKV	200	25	50	100	200
15	UWP	200	25	50	100	200
25	V1	200	25	50	100	200
35	REV	200	25	50	100	200

- Gym

			Tuition with Owned Number	
Tile	Gym	Purchase Cost	1	2
12	PAC	150	4 * dice sum	10 * dice sum
28	CIF	150	4 * dice sum	10 * dice sum

5. Unownable property: a property which a player cannot buy/ trade/ improve/ mortgage/ unmortgage but should complete the action depending on the tile: Collect OSAP, DC Tims Line, Go to Tims, Goose Nesting, Tuition, Coop fee, SLC, Needles Hall.

To achieve high cohesion and low coupling, individual concrete classes are implemented for Grid, Player, Dice, academic building, residence, gym and each of the unownable properties, several design patterns are also used and will be discussed later.

Design

Initially, the Watopoly Controller class is implemented to handle player commands and interact with the game's kernel to perform various functionalities such as initializing, playing, loading, saving, testing, and printing the game. This class is composed of two essential classes: Grid and Player, both of which offer various functionalities necessary for the game.

The Grid class represents the game board and offers methods for board creation, initialization with different property types, and interaction with the game board during

gameplay. The Property objects stored in the map vector represent various spaces on the game board, including academic buildings, residences, and other buildings located on the Uwaterloo campus. This class lays the foundation for implementing Watopoly game mechanics.

The Player class represents the game's players and includes variables to track their cash, owned properties, and other attributes. This class provides player actions such as movement, buying and selling properties, paying tuition, mortgaging properties, and trading with other players. It also enables players to view their assets and take actions related to improving or degrading their properties. The Player class plays a crucial role in the game's implementation as it manages the actions and assets of individual players within the game.

The Property class is attached to the Grid class and has a composition relationship with it. This class represents the properties/buildings in the game and includes variables to track their attributes such as name, type, ownable, and movable. This class offers functions such as getting and setting attributes, neighbors, guests, monopoly sets, and information related to tuition, movement instructions, and event times.

To handle the one-to-many relationship between the building and its operations such as purchasing, improving, and paying tuition, the observer design pattern is employed. This involves implementing a Subject class and an Observer class, both of which are derived from the Building class. The Building class also has two other derived classes, Unownable Building and Ownable Building, which categorize all the buildings on campus. The Ownable Building class allows for purchasing, improving,

and setting the building owner, among other actions. To enable these actions, the observer for the Ownable Building class is set to include other Ownable Buildings within the same category as well as the player.

The Ownable class contains additional data members and functions related to properties that can be owned, such as the owner of the property, the purchase cost, and whether the property can be improved. The class also support functionalities allow for properties to be improved and degenerated, and for players to check the improvement level of a property and whether other properties in the same monopoly set have been improved.

Unownable is a derived class of the Property class that represents properties that cannot be owned by players in the game. It provides functionalities such as calculating tuition payment, providing movement instructions for players, getting the guest list of the property, and getting the improvement level of the building. Unlike the Ownable class, Unownable properties cannot be bought or owned by any player in the game.

After the basic structure of the program is built, more classes are added to the Ownable and Unownable classes, each is responsible for one specific functionalities in one square. The derived classes of Ownable, including Academic, Residence, and Gym, override improvement related functionalities for academic buildings, also add functionalities of rent paying for residences and gym costs. On the other hand, the derived classes of Unownable, such as GoTims, OSAP, Tuition, GooseNesting, SLC, NeedlesHall, etc. provide functionalities specific to the various unownable spaces on the game board, such as random events, penalty payments, or movement instructions.

To achieve special events like the Roll Up the Rim Cup inherited from the Unownable class, the decorator design pattern is used. This enables the program to dynamically select a suite event when the player is in a special location, such as SLC or Needles Hall. The decorator design pattern also ensures future optimization and better code readability for other programmers.

Overall, out design of the game follows the object-oriented programming paradigm, with various classes representing different aspects of the game. The controller is responsible for managing player input and communicating with the game's core to carry out different operations. The Gird manages the board and the positions of the players on the board, while the Player class represents each player in the game, controls the actions of each player in the game. The Property class represents all squares in the game, and it is further divided into Ownable and Unownable subclasses, with specific properties and functionalities each.

Resilience to change

Our design is well-suited to accommodate change due to its use of object-oriented programming principles. The design is modular and each class is responsible for specific functionalities, making it easy to add or remove functionality as needed. For example, if new properties or features are added to the game, new classes can be easily added to represent them without affecting existing classes. Additionally, inheritance and polymorphism allow for the easy addition of new functionality without changing existing code.

Coupling is a measure of the interdependence between classes, and the design shows low coupling between classes. The classes are designed to be independent and have minimal dependencies on other classes. For example, the Grid class interacts with the Player and Property classes but is not tightly coupled to either of them. Similarly, the Property class is not tightly coupled to the Grid class, and the Player class interacts with both but is not dependent on either.

Cohesion refers to the degree to which the responsibilities of a class are related and the design exhibits high cohesion between classes. Each class has a well-defined set of responsibilities that relate to its specific functionality, and each class has a clear, single purpose. For example, the Player class is responsible for managing player actions and assets, while the Grid class is responsible for managing the game board. This high cohesion means that each class can be easily understood and modified without affecting other parts of the program.

In conclusion, the design of the Watopoly game follows object-oriented programming principles and exhibits low coupling and high cohesion between classes, making it easy to accommodate change and modify specific parts of the program without affecting the entire system. The use of inheritance and polymorphism allows for easy addition of new functionalities and the classes are well-defined with specific purposes, making them easy to understand and modify.

Extra Credit Features

To begin with, our program includes a fixed seed feature, which ensures consistent

results each time the game is played. We have also implemented a load and save feature, allowing players to save their progress and resume later. Furthermore, we added a "quit" command, allowing players to exit the game at any point. Our program also has the capability to add an extra player beyond the original set. Most importantly, we have designed our program to be free of memory leaks. To achieve this, all pointers that store data are shared pointers.

Answer to Questions

1. The observer pattern is well-suited for this game, as it allows the Player and Building classes to inherit from the abstract Observer class, and the Building class to inherit from the abstract Subject class. Frist of all, this enables ownable properties to notify players who own buildings, as well as other ownable buildings of the same type, which is useful for determining whether buildings can be improved or traded to other players, as well as notifies all other buildings(observers) in its monopoly. Secondly, the OwnableBuilding class is a subject that notifies its observers when its status changes or certain events occur. Similarly, the UnownableBuilding class is a child of both the Observer and Subject classes, which allows players to know the status and number of active Roll Up the Rim cups in real-time. This enables each player to make informed decisions about their choices, as the total number of cups owned by all players cannot exceed four.

2. We believe that the decorator pattern will make SLC and Needles Hall more flexible, as it allows for the creation of both actual and additional models, such as the

Roll Up the Rim Cup. When landing on either of these buildings, there is a rare chance (1%) of receiving a winning Roll Up the Rim cup, which allows the player to bypass the DC Tims line for free. However, at any given time, there cannot be more than four active cups (meaning that if four cups are active, there is a 0% chance of receiving another one). To handle these events, we would like to implement the decorator pattern. This allows the program to update the status of events as needed and to check the next event if the previous one does not apply. Additionally, this approach can facilitate the addition of new events in the future by implementing them as decorators, which will decrease the modification needed and improve the readability of the program.

3. The use of decorator pattern for the implementation of improvements is not recommended due to the high complexity of the different situations that can arise. Although using decorator pattern would add flexibility to our program and adding functionality would be easier in the case, there are about 132 integers related to the tuition with improvement, which makes it difficult to implement using the decorator pattern. Furthermore, some ownable buildings in the same monopoly may have different tuition amounts with the same level of improvements, making it harder to build the data with the decorator pattern. If we want to use decorator pattern, we would need to implement 5 concrete decorator classes, each for 1 level of improvement, this would be tedious and unnecessary since we can implement this through if statements and methods. In summary, building a whole design pattern for this simple feature is not worth the effort.

Final Question

1. **Lessons learned about developing software in teams:**

This project taught us the importance of teamwork and communication when developing software in a team. We learned that it is crucial to have a shared understanding of the project's objectives and requirements, as well as a clear plan for dividing the work among team members. We also learned that it is important to stay punctual and focused, as a delay or mistake by one team member can impact the entire project's progress. Additionally, we learned that careful attention to detail and rigorous testing are essential for avoiding errors and ensuring the project's success.

2. **What would have been done differently:**

If we had the chance to start over, in addition to the steps mentioned earlier, we would have also emphasized early and frequent testing. This would have helped us catch and fix errors earlier in the development process, reducing the amount of time and effort required for testing and bug fixing later on. We would have also made sure to document our code clearly and thoroughly, making it easier for team members to understand and maintain the codebase. Finally, we would have taken advantage of available tools and technologies, such as version control and automated testing frameworks, to streamline the development process and improve code quality.

Conclusion

In conclusion, the experience of creating Watopoly has been an incredible learning

opportunity for our team. We are immensely grateful for the chance to work together and develop a game that we can all be proud of. Through the process, we learned valuable skills, such as effective planning, task allocation, and communication, which will undoubtedly serve us well in our future endeavors. As a team, we faced numerous challenges, but we persevered and supported each other throughout, resulting in a successful project. What initially seemed like an insurmountable task became a reality through our hard work and dedication. We are truly proud of what we accomplished as a team and look forward to applying the lessons we learned to our future work and studies. Overall, creating Watopoly was a valuable experience that we will always remember as a testament to our teamwork, dedication, and perseverance.