Coded by Nick Barpoulis, Sienna Mosher, Emma Overly, Anushikha Sharma

Background/Plan

- Based on the iconic game of Monopoly
    - Our version focuses on buying Bucknell housing properties
- Our goal was to code a realistic and interactive game of Monopoly with four players
- We broke this project up into two phases: the functionality (Sprint 1) and the GUI (Sprint 2)

Scrum Methodology- User Stories

“As a customer, I would like the player to move around the board and interact with the spots on the board”

“As a customer, I would like to have a bank account so I can manage it”

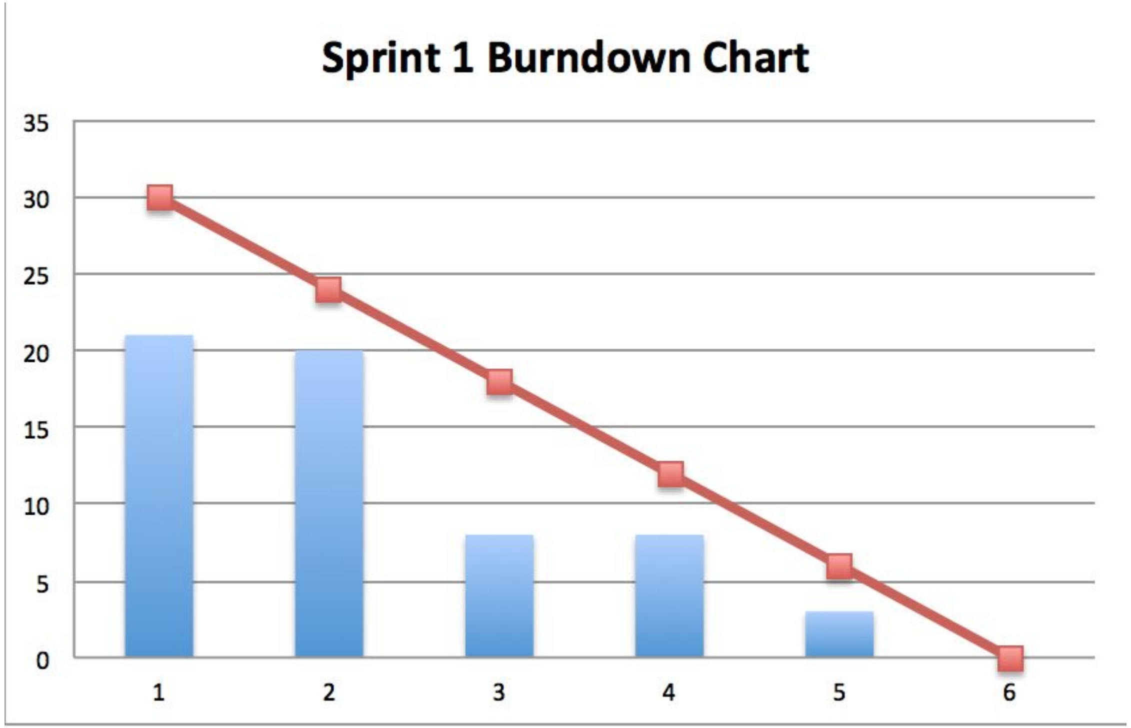
“As a customer, I would like to have my turn be interactive, roll dice, move space, show consequence”

“As a customer, I would like to be able to see my playing piece move around the board”

Scrum Methodology

Strengths	Weaknesses
- Organizes our priorities based on the user stories and then the task boards - The task boards also held us accountable for getting work to be completed in a timely manner - Separating the work into sprints, allowed us to finish all the necessary functionality in Sprint 1, to make it easier to code the GUI in Sprint 2	- Scrum would work really well in a work environment, but it isn't perfect for students- we work in spurts rather than consistent day-to-day work - Difficult to predict how long each task will take to complete-- sometimes skewing our burndown charts

Burndown Chart of Sprint 1



Design and Implementation

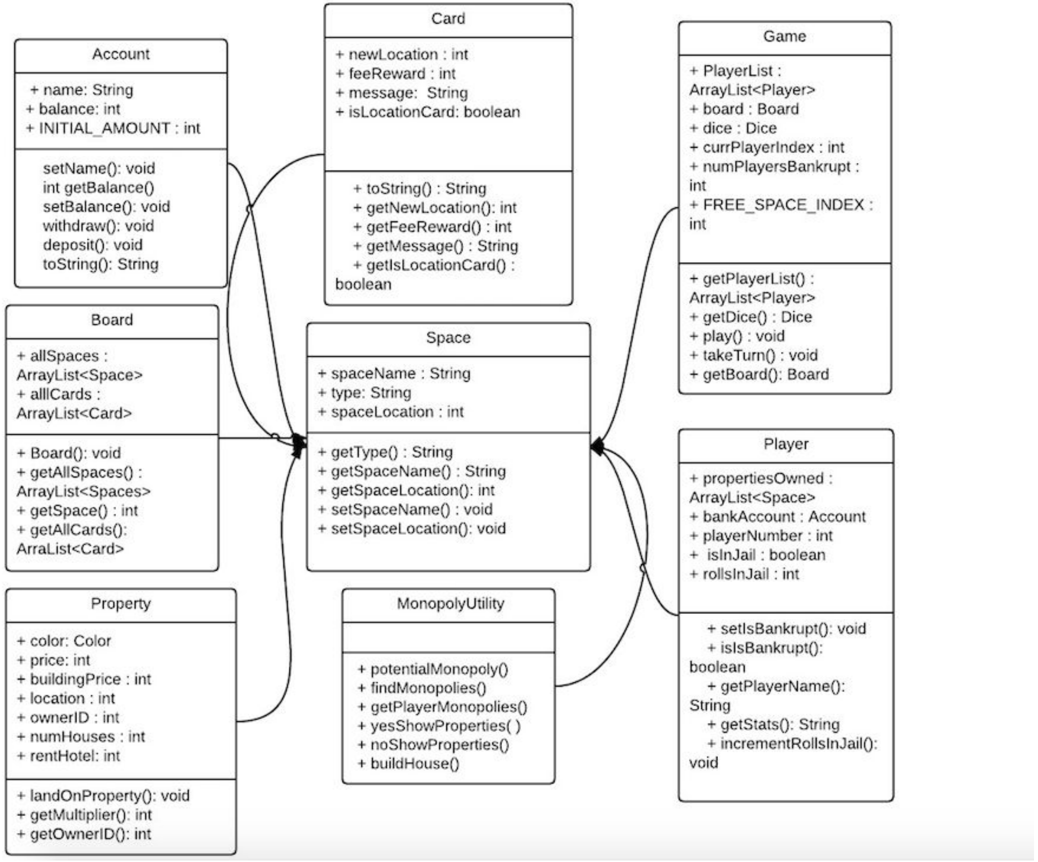
Major classes: Space parent class, Property subclass of Space, Community Chest subclass of space (and so on with all other spaces on the monopoly board)

All functionality was controlled in the Game class: all Players, the board, and dice were instantiated within this class.

The players keep track of where they are on the board, move accordingly, and thus react to the Space object that corresponds to their location on the board, extracting the functionality from the type of subclass the space is

Using MVC, we created an instance of the Game class within the model, and were able to interact with the Game from the Controller, thus updating the View

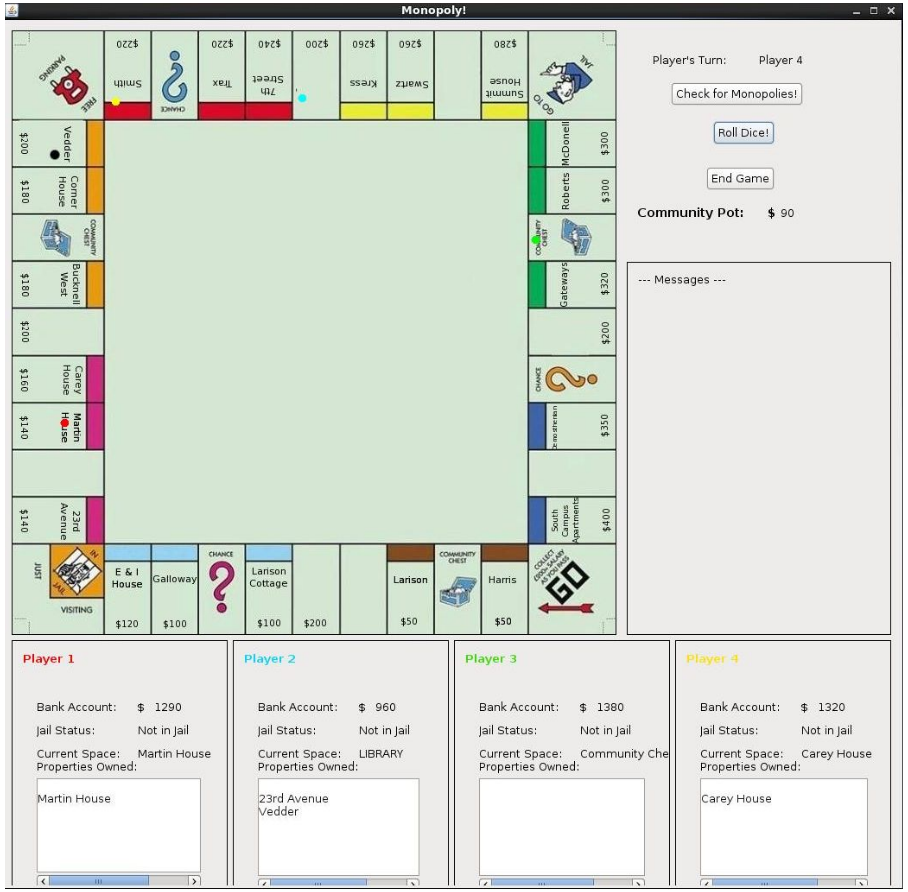
Basic UML Overview



Data Structures/Design Elements

- We used a lot of ArrayLists and Arrays to keep track of the spaces on the board, the properties a player owns, the list of players in the game
- Enums for property colors
- The methods that were the core functionality of the game (landing on a property ➜ being prompted to purchase it) required us to be careful with OOP
  - Kept track of making sure we were updating the correct instance of an object
- We utilized MVC in updating all of the GUI as the game played out, keeping the game consistent.

User Interface



Player's Turn: Player 4

Check for Monopolies!

Roll Dice!

End Game

Community Pot: $ 90

--- Messages ---

Player 1

Bank Account: $ 1290
Jail Status: Not in Jail
Current Space: Martin House
Properties Owned:
Martin House

Player 2

Bank Account: $ 960
Jail Status: Not in Jail
Current Space: LIBRARY
Properties Owned:
23rd Avenue
Vedder

Player 3

Bank Account: $ 1380
Jail Status: Not in Jail
Current Space: Community Che
Properties Owned:

Player 4

Bank Account: $ 1320
Jail Status: Not in Jail
Current Space: Carey House
Properties Owned:
Carey House

Challenges/Comments

- It was difficult to continuously update the CRC cards and UML with the refactoring of the code.
    - The code was refactored quite a lot over the process, and this led to many changes
- We did feel like there was a bit of a time constraint: our main goal was to have nearly all Monopoly functionality, and because of that, less time was given for the GUI
- Difficult to stay with the Scrum methodology: it’s difficult for a student to predict when they can work on a project, not a consistent burndown chart

Further Work

- Mortgaging functionality
- Taking a second turn after rolling doubles
- Improving the user interface:
    - Would like to have more realistic “cards” rather than information popup windows
    - A more comprehensive list of the properties the players own
        - Possibly separated by color, property price, show the rent etc.
        - Will allow for users to check for monopolies in a more efficient way
    - Having the users choose Bucknell-themes playing pieces (such as a bison)
    - Play music for the opening of the game