Sprint 10 Documentation

Summary Data

Team Number: 13
Team Lead: Ankeet
Sprint Start: 08/05/2020
Sprint End: 15/05/2020

Individual Key Contributions

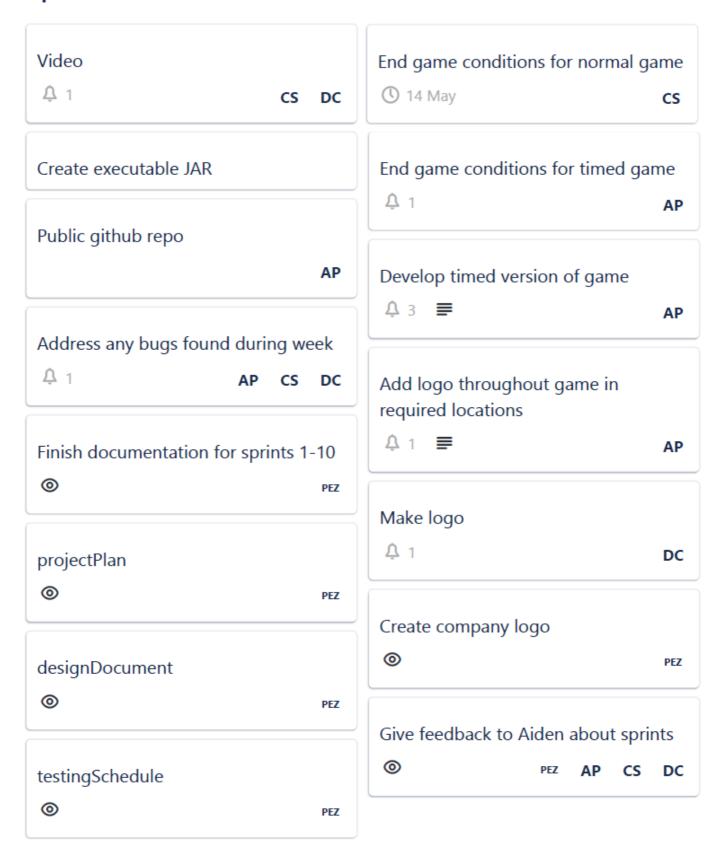
Team Member	Key Contributions
Aiden	Documentation
Ankeet	Implementation
Chris	Implementation
Duarte	Implementation

Task Cards

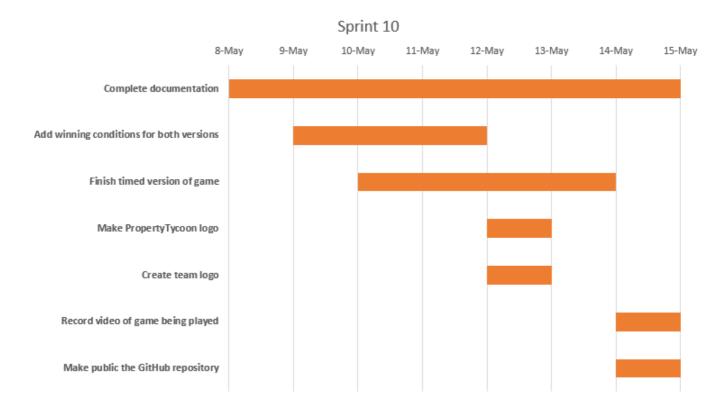
- Complete documentation
- Finish timed version of normal game
- Record video of game
- Create an executable JAR file
- Make public the GitHub repo
- · Add winning conditions for both versions of the game
- Make PropertyTycoon logo
- Create company logo

The image below shows the tasks set out on Trello during our weekly meeting

Sprint 10



Gantt Chart



Requirements Analysis

Functional Requirements

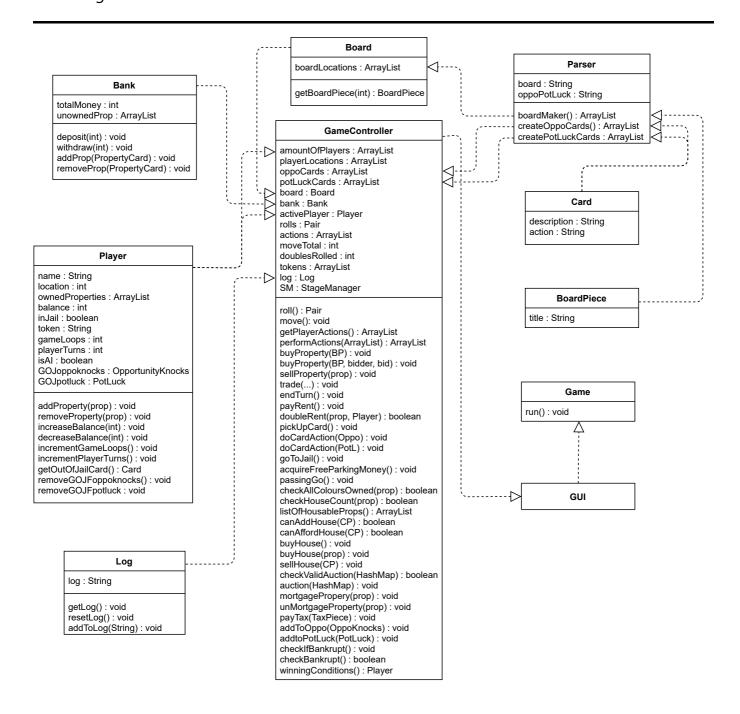
- F1
- The software shall have a timed version of the game. There shall exist a timer to count down from a set amount of time.
- F2
- The software shall end a game considering what type of game is being played.
- F3
- A logo should be developed to associate the game of Property Tycoon.

Non-Functional Requirements

- NF1
 - For the timed game, a timer is preset for a determined amount of time. When the timer stops, the software shall wait for every player to have equal turns in the game. Once this has been detected, the player with the largest amount of assets wins the game. To calculate a player's assets, the software shall sum the balance, the price of owned properties and the price of all houses on all owned properties.
- NF2
 - The winning conditions for the normal game of Property Tycoon shall be when there is a single player exist in the game. With this, all other players have become bankrupt.

Design

UML Diagram



GameController class structure

GameController

amountOfPlayers : ArrayList playerLocations : ArrayList oppoCards : ArrayList potLuckCards : ArrayList

board : Board bank : Bank activePlayer : Player

rolls : Pair actions : ArrayList moveTotal : int doublesRolled : int tokens : ArrayList log : Log

SM : StageManager

roll(): Pair move(): void

getPlayerActions(): ArrayList performActions(ArrayList): ArrayList

buyProperty(BP): void

buyProperty(BP, bidder, bid) : void

sellProperty(prop) : void

trade(...) : void endTurn() : void payRent() : void

doubleRent(prop, Player) : boolean

pickUpCard() : void doCardAction(Oppo) : void doCardAction(PotL) : void

goToJail(): void

acquireFreeParkingMoney(): void

passingGo(): void

checkAllColoursOwned(prop): boolean checkHouseCount(prop): boolean listOfHousableProps(): ArrayList canAddHouse(CP): boolean canAffordHouse(CP): boolean

buyHouse(): void buyHouse(prop): void sellHouse(CP): void

checkValidAuction(HashMap): boolean

auction(HashMap): void
mortgagePropery(prop): void
unMortgageProperty(prop): void
payTax(TaxPiece): void
addToOppo(OppoKnocks): void
addtoPotLuck(PotLuck): void
checkIfBankrupt(): void
checkBankrupt(): boolean
winningConditions(): Player

Timed

amountOfPlayers : ArrayList playerLocations : ArrayList oppoCards : ArrayList potLuckCards : ArrayList

board : Board bank : Bank activePlayer : Player

rolls : Pair actions : ArrayList moveTotal : int doublesRolled : int tokens : ArrayList

log: Log

SM: StageManager

+t : Timer +task : TimerTask

roll(): Pair move(): void

getPlayerActions(): ArrayList performActions(ArrayList): ArrayList

buyProperty(BP): void

buyProperty(BP, bidder, bid): void

sellProperty(prop): void

trade(...) : void
endTurn() : void
payRent() : void

doubleRent(prop, Player): boolean

pickUpCard(): void doCardAction(Oppo): void doCardAction(PotL): void

goToJail(): void

acquireFreeParkingMoney(): void

passingGo(): void

checkAllColoursOwned(prop): boolean checkHouseCount(prop): boolean listOfHousableProps(): ArrayList canAddHouse(CP): boolean canAffordHouse(CP): boolean

buyHouse(): void buyHouse(prop): void sellHouse(CP): void

checkValidAuction(HashMap): boolean

auction(HashMap): void mortgagePropery(prop): void unMortgageProperty(prop): void payTax(TaxPiece): void addToOppo(OppoKnocks): void

addtoPotLuck(PotLuck): void checkIfBankrupt(): void checkBankrupt(): boolean +fullTurn(): boolean +winningConditions(): Player

5/7

Player class structure

Player

name : String location : int

ownedProperties : ArrayList

balance: int inJail: boolean token: String gameLoops: int playerTurns: int isAI: boolean

addProperty(prop): void removeProperty(prop): void increaseBalance(int): void decreaseBalance(int): void incrementGameLoops(): void incrementPlayerTurns(): void

AiPlayer

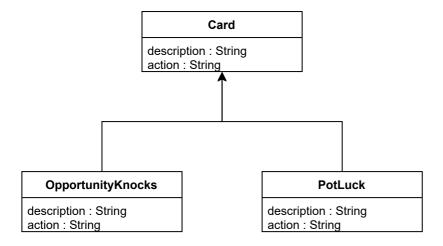
name : String location : int

ownedProperties : ArrayList

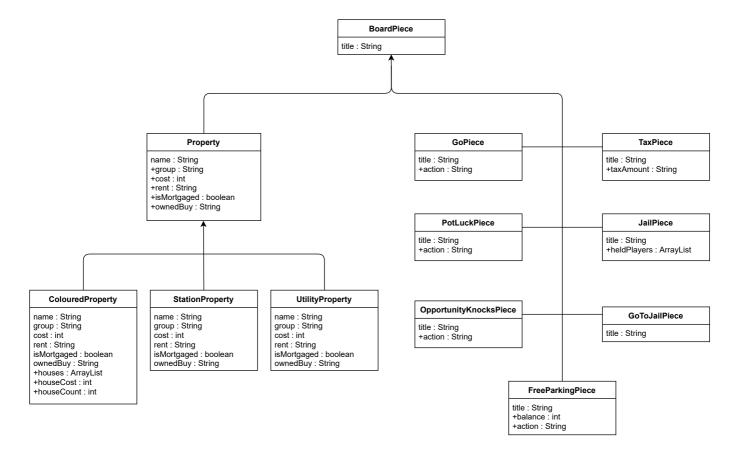
balance: int inJail: boolean token: String gameLoops: int playerTurns: int isAl: boolean

addProperty(prop): void removeProperty(prop): void increaseBalance(int): void decreaseBalance(int): void incrementGameLoops(): void incrementPlayerTurns(): void +DoesAiBuy(): boolean +AiAuctionValue(prop): int +tryBuyHouse(): boolean +trySellProperty(): boolean

Card class structure



BoardPiece class structure



Test Plan

All the testing done in this sprint is system-wide testing to ensure the Al players can operate properly and perform actions just like any other real life player. System-wide testing was also being used to test the timed version of the game.

Summary of Sprint

This sprint was the final sprint of the project. The sprint was mainly the preparation of the documentation and the final release of the game.