Sprint 6 Documentation

Summary Data

- **Team Number:** 13
- **Team Lead:** Ankeet
- **Sprint Start:** 09/03/2020
- **Sprint End:** 06/04/2020

Note: this sprint was extended to a month due to special circumstances

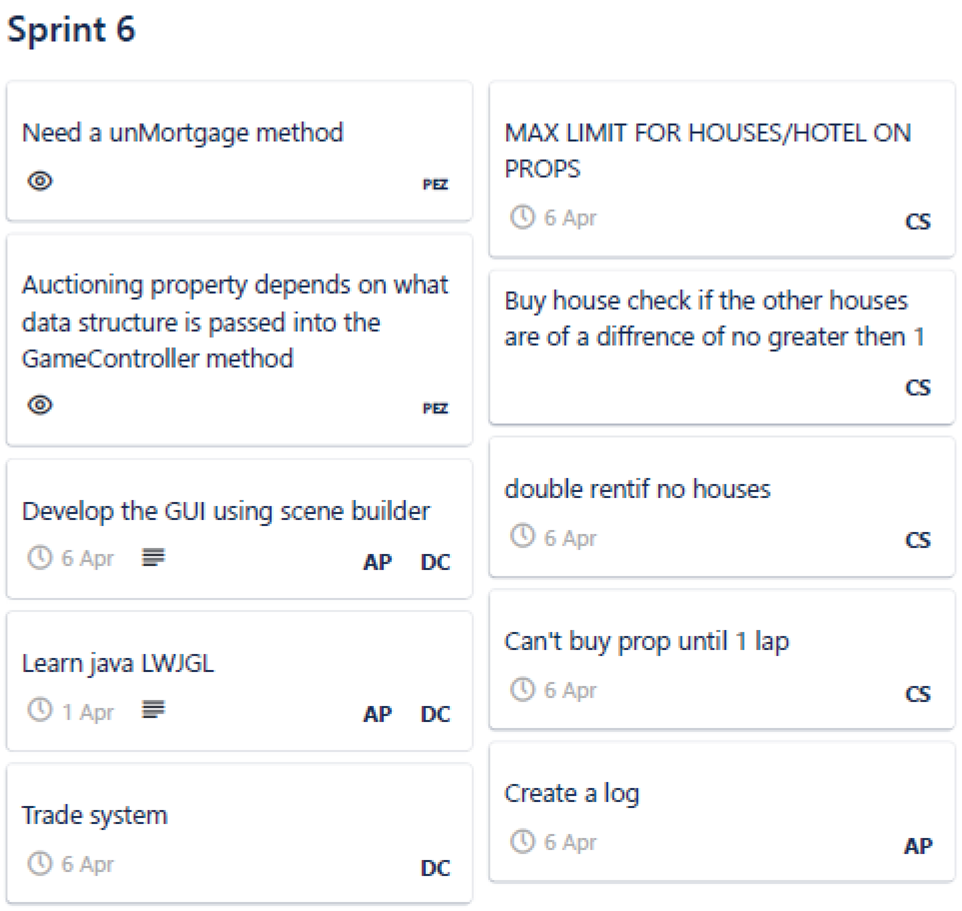
Individual Key Contributions

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

Task Cards

- unMortgage method
- Decide on data structure to pass players and bids from GUI to GameController
- Develop GUI using scene builder
- Implement a trading system
- Implement the maximum number of houses on a property
- Add houseChecks into buyHouse
- Double rent if all properties of a group are owned with no houses upon them
- Property cannot be purchased until the player has looped around the board at least once
- Show the log in the GUI

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



Sprint 6

Need a unMortgage method
PEZ

Auctioning property depends on what data structure is passed into the GameController method
PEZ

Develop the GUI using scene builder
6 Apr
AP DC

Learn java LWJGL
1 Apr
AP DC

Trade system
6 Apr
DC

MAX LIMIT FOR HOUSES/HOTEL ON PROPS
6 Apr
CS

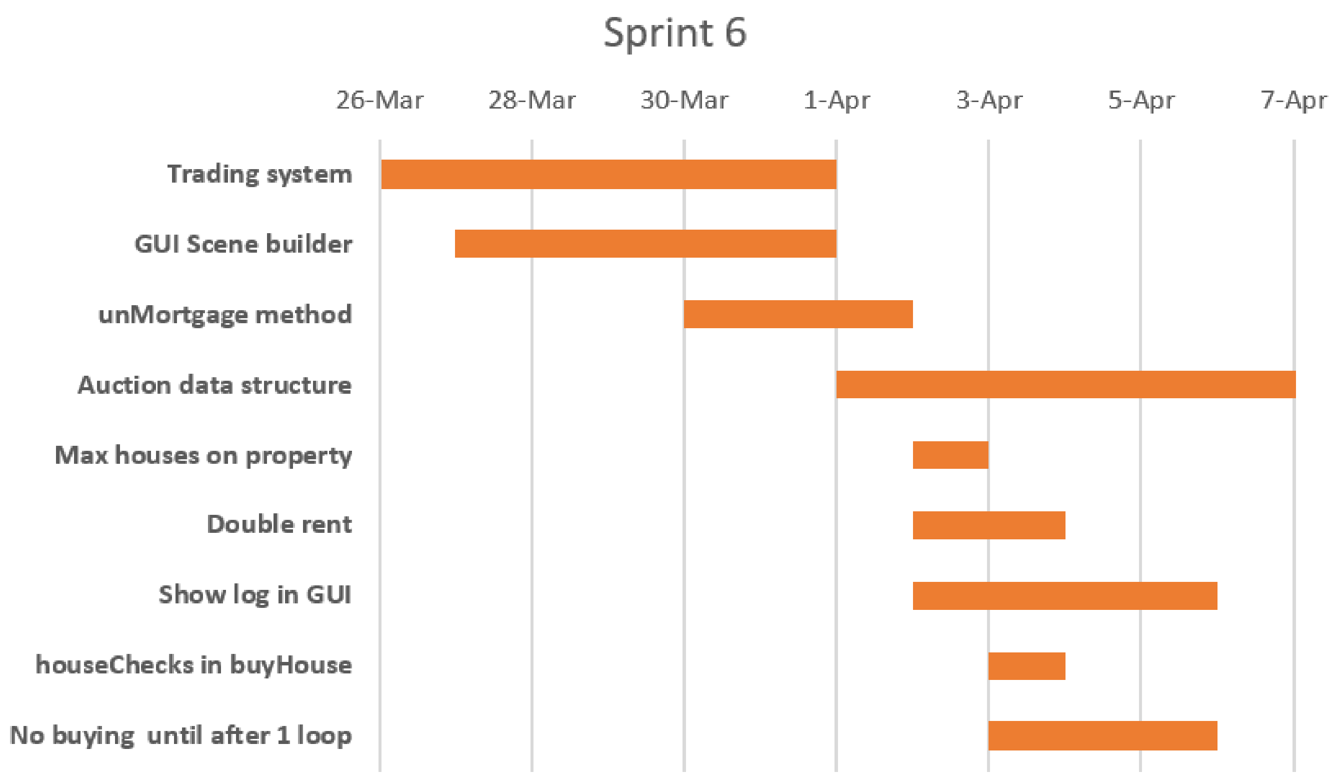
Buy house check if the other houses are of a diffrence of no greater then 1
CS

double rentif no houses
6 Apr
CS

Can't buy prop until 1 lap
6 Apr
CS

Create a log
6 Apr
AP

Gantt Chart



Requirements Analysis

Functional Requirements

- F1
  - Players shall have the ability to unmortgage a property that is currently mortgaged
- F2
  - The software shall implement a trading system between players
- F3
  - The software shall increase the rent across a colour group if a player owns all of the colour group and has no houses on them
- F4
  - Players cannot purchase any property until they have travelled at least one loop around the board
- F5
  - The software should display a log of previous events

Non-Functional Requirements

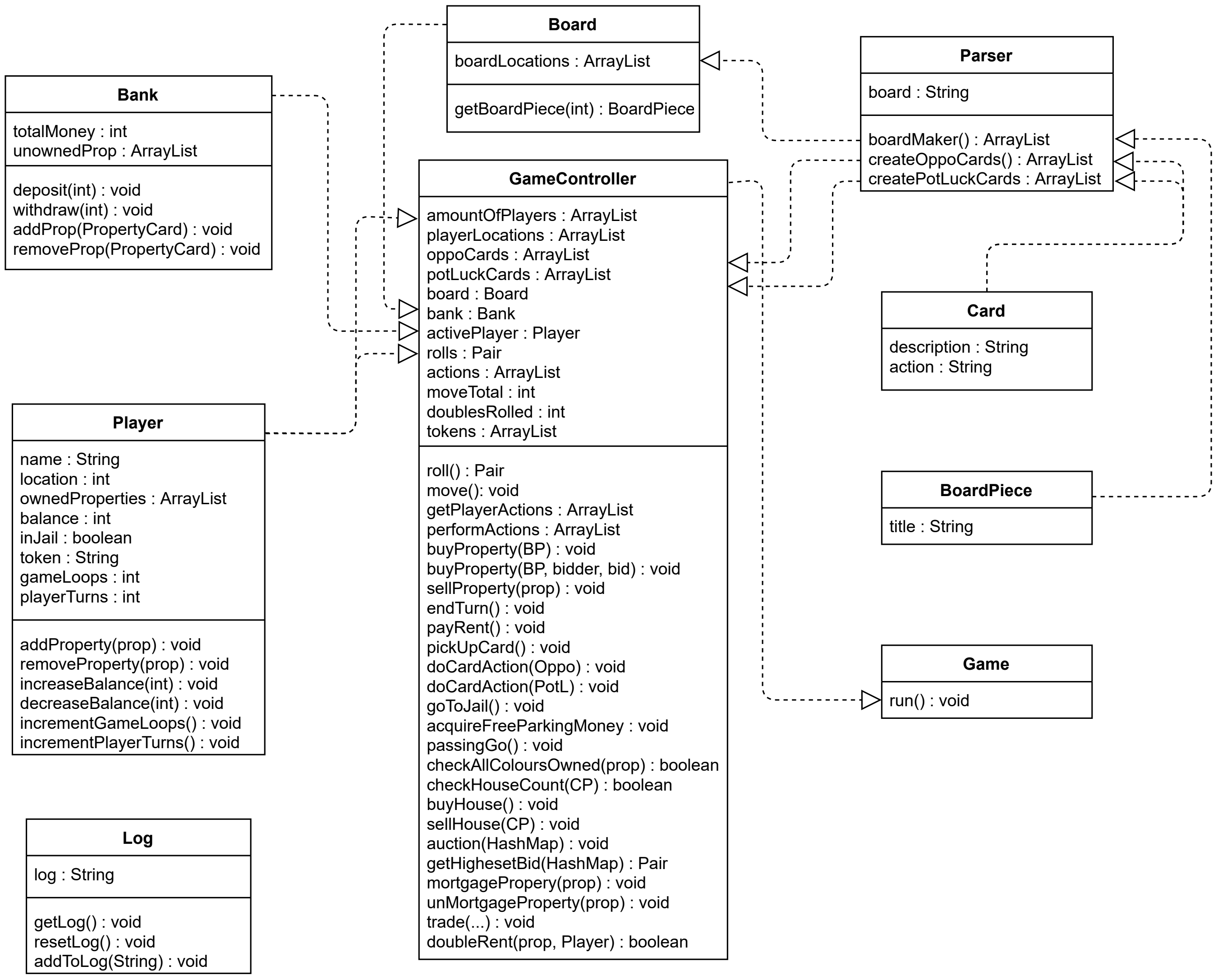
- NF1
  - When a player is purchasing a house, the software shall check if the property can have an additional house
- NF2
  - To unmortgage a property, the player must pay half the purchasing price of the property to the bank
  - The property can then start taking rent from other players if they land on the property
- NF3
  - When a trade is created, it must be accepted by the receiver to be a successfull trade. If rejected by the receiver, the trade is dismissed.

Domain Requirements

- D1
  - The team must ask the client what the maximum number of houses that can be bought on a single property

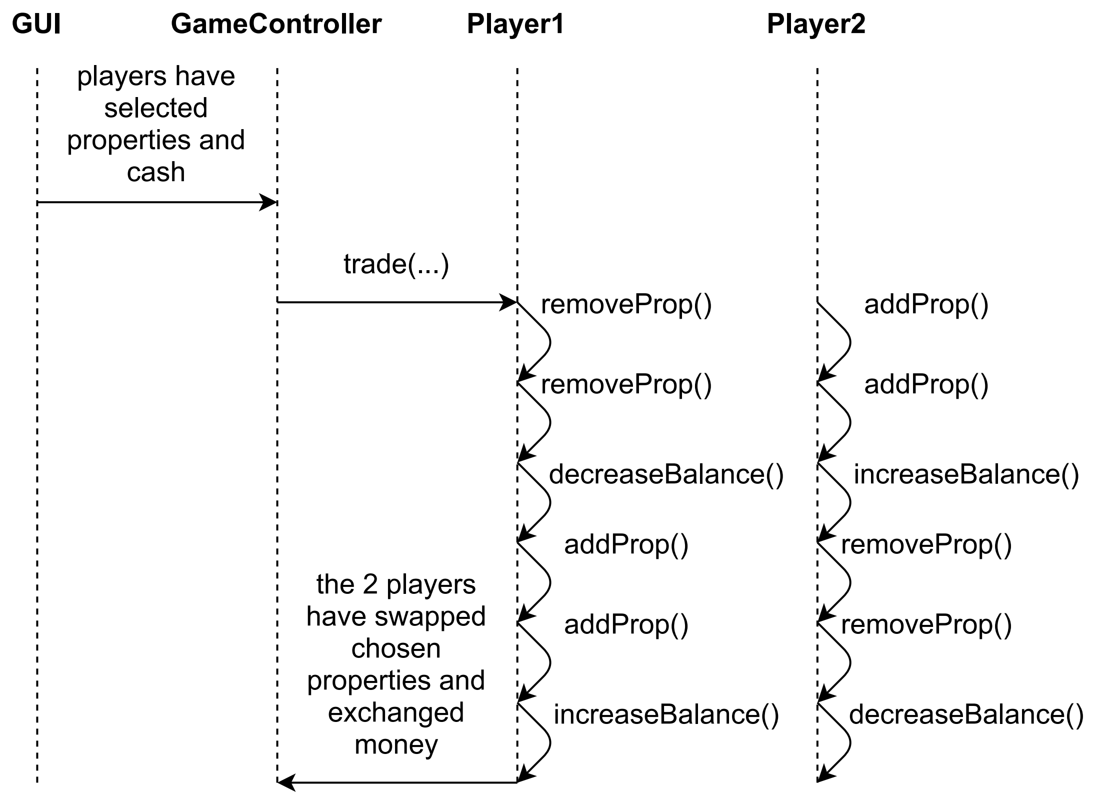
Design

UML Diagram

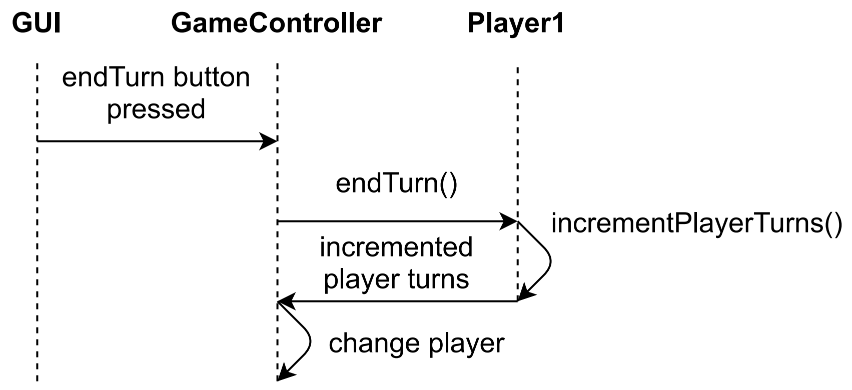


Sequence Diagrams

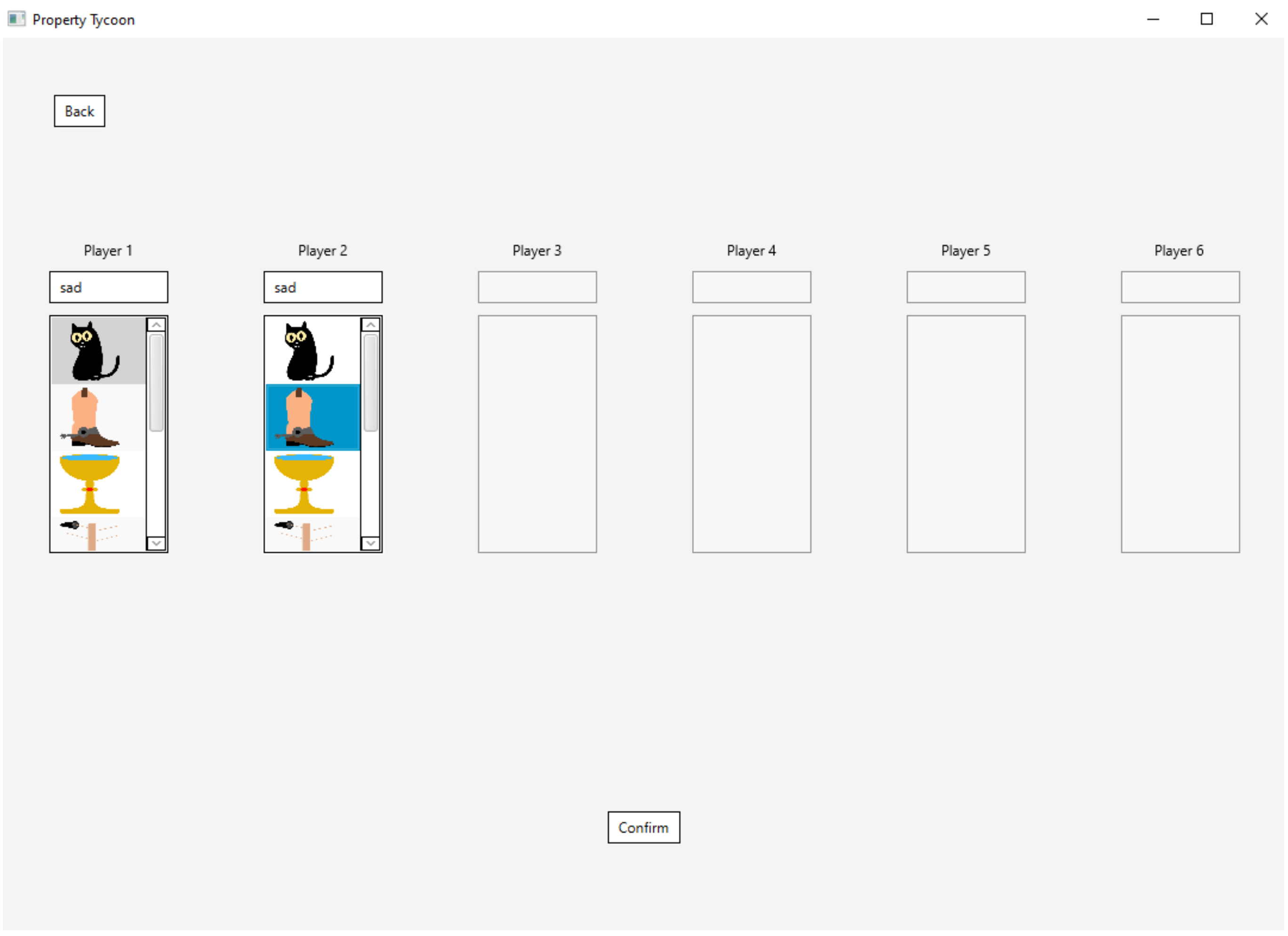
Trading mechanic between 2 players



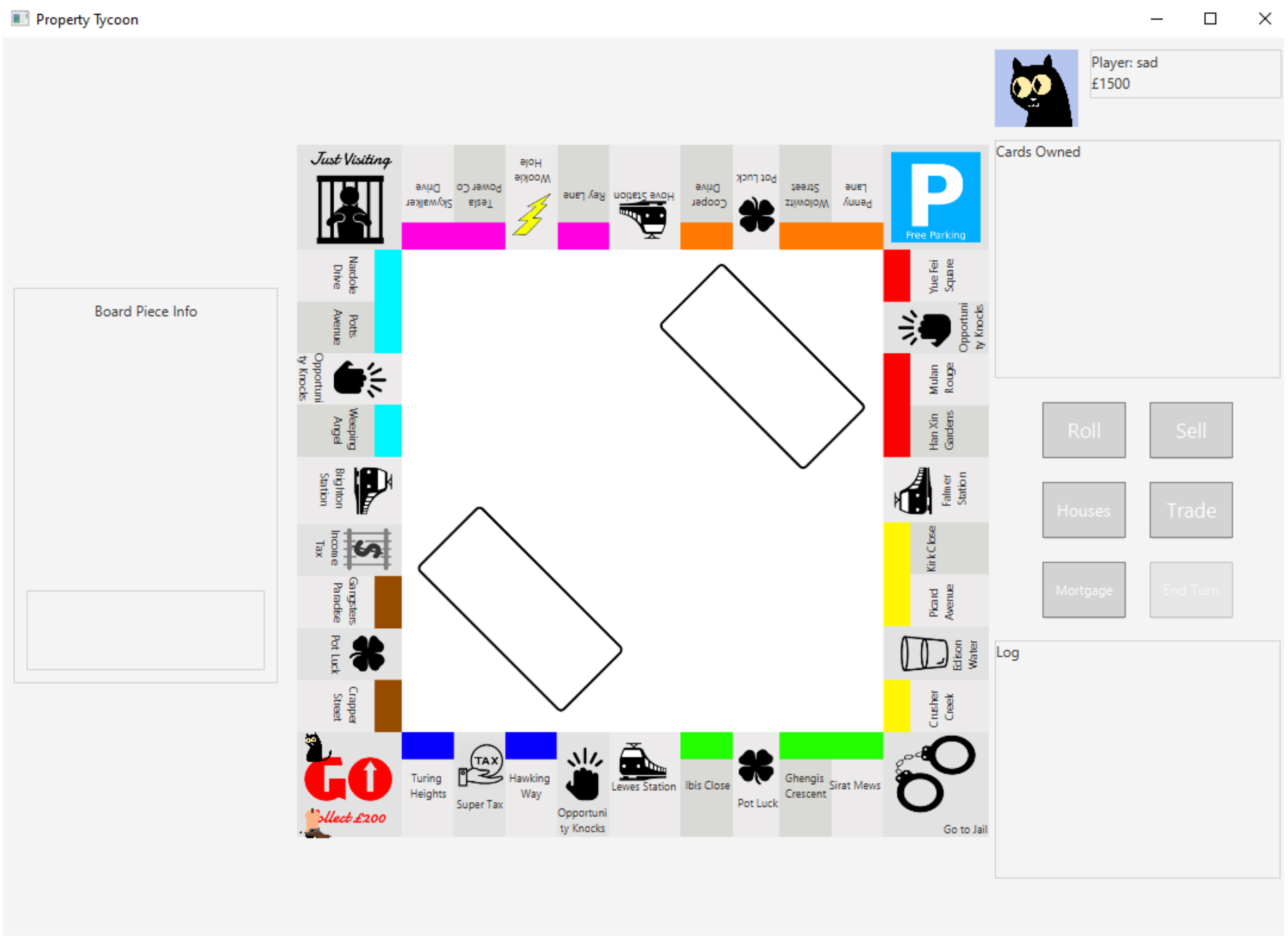
Ending a player's turn



User Interface



An update to last sprint on choosing of names and tokens of players was that instead of words, we have added what the token would look like.



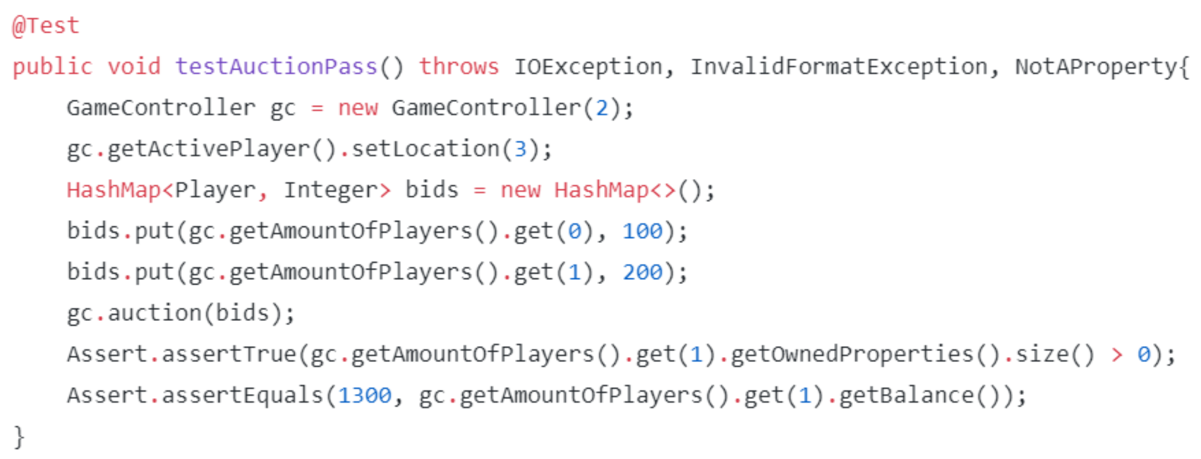
An addition to the main user interface is that the left side of the board is now a box highlighting a specific board piece on the board. This has not yet been implemented, but the appropriate space is built for it.

Test Plan

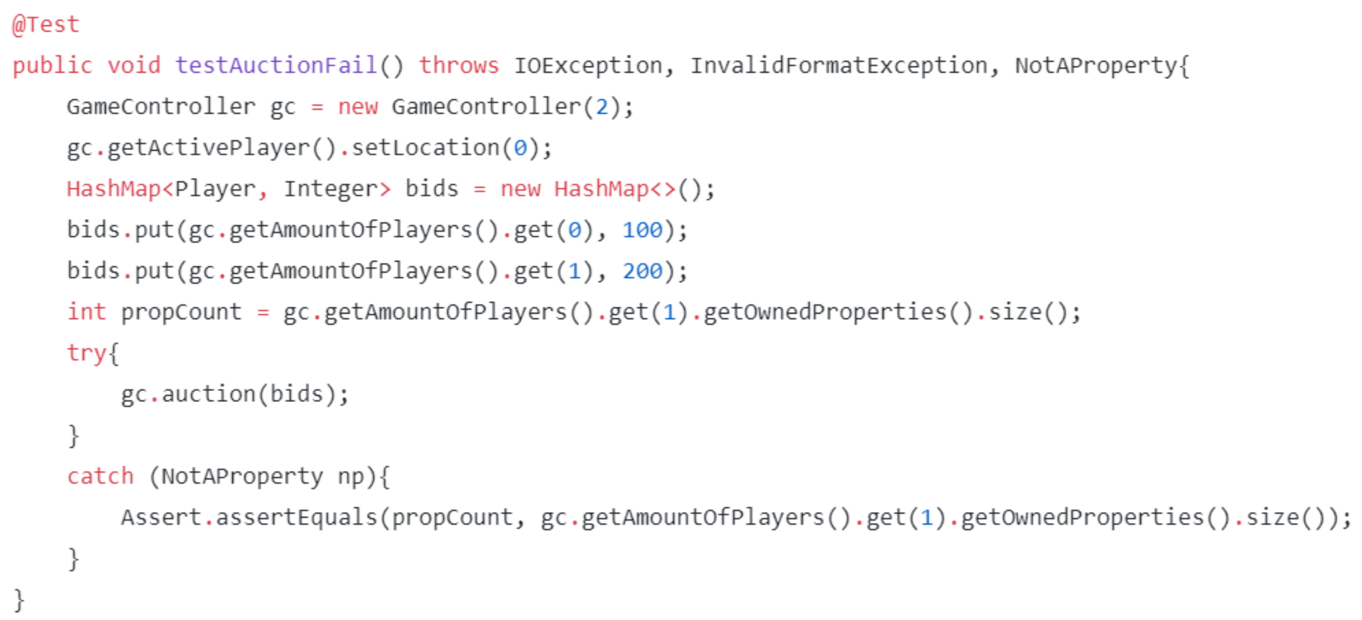
The main updates in terms of testing is with:

1. Auctioning
2. Mortgaging
3. Rent

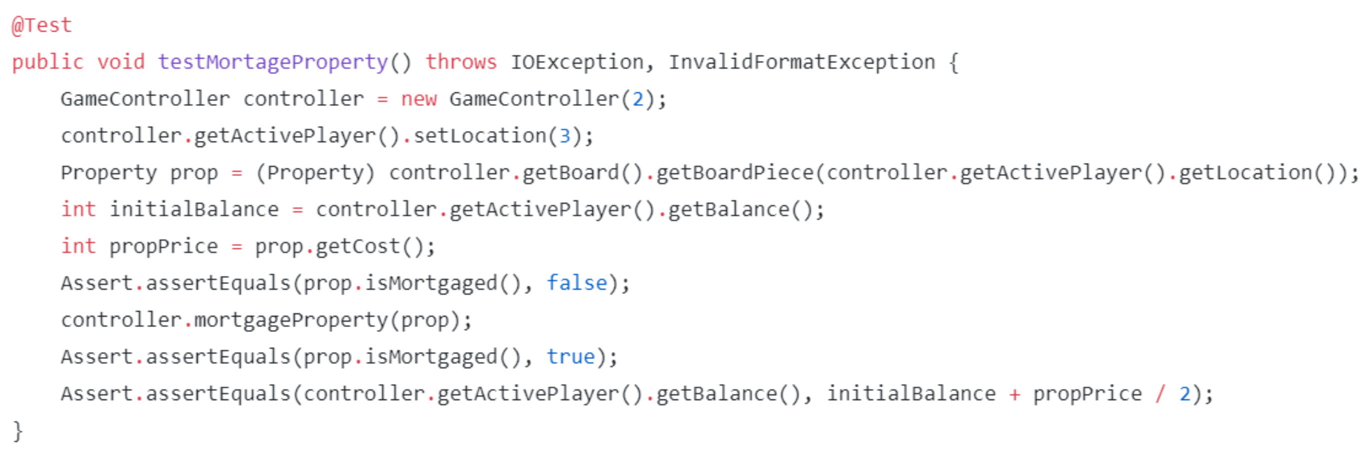
The next 3 images below highlight the testing capabilities of ensuring the methods function properly



```
@Test
public void testAuctionPass() throws IOException, InvalidFormatException, NotAProperty{
    GameController gc = new GameController(2);
    gc.getActivePlayer().setLocation(3);
    HashMap<Player, Integer> bids = new HashMap<>();
    bids.put(gc.getAmountOfPlayers().get(0), 100);
    bids.put(gc.getAmountOfPlayers().get(1), 200);
    gc.auction(bids);
    Assert.assertTrue(gc.getAmountOfPlayers().get(1).getOwnedProperties().size() > 0);
    Assert.assertEquals(1300, gc.getAmountOfPlayers().get(1).getBalance());
}
```



```
@Test
public void testAuctionFail() throws IOException, InvalidFormatException, NotAProperty{
    GameController gc = new GameController(2);
    gc.getActivePlayer().setLocation(0);
    HashMap<Player, Integer> bids = new HashMap<>();
    bids.put(gc.getAmountOfPlayers().get(0), 100);
    bids.put(gc.getAmountOfPlayers().get(1), 200);
    int propCount = gc.getAmountOfPlayers().get(1).getOwnedProperties().size();
    try{
        gc.auction(bids);
    }
    catch (NotAProperty np){
        Assert.assertEquals(propCount, gc.getAmountOfPlayers().get(1).getOwnedProperties().size());
    }
}
```



```
@Test
public void testMortagePropery() throws IOException, InvalidFormatException {
    GameController controller = new GameController(2);
    controller.getActivePlayer().setLocation(3);
    Property prop = (Property) controller.getBoard().getBoardPiece(controller.getActivePlayer().getLocation());
    int initialBalance = controller.getActivePlayer().getBalance();
    int propPrice = prop.getCost();
    Assert.assertEquals(prop.isMortgaged(), false);
    controller.mortgageProperty(prop);
    Assert.assertEquals(prop.isMortgaged(), true);
    Assert.assertEquals(controller.getActivePlayer().getBalance(), initialBalance + propPrice / 2);
}
```

```java
@Test
public void payDoubleRent() throws IOException, InvalidFormatException, NotAProperty {
    GameController controller = new GameController(2);
    controller.getActivePlayer().setLocation(1);
    controller.buyProperty(controller.getBoard().getBoardLocations().get(controller.getActivePlayer().getLocation()));
    controller.getActivePlayer().setLocation(3);
    controller.buyProperty(controller.getBoard().getBoardLocations().get(controller.getActivePlayer().getLocation()));

    controller.endTurn();
    controller.getActivePlayer().setLocation(3);
    controller.performActions(controller.getPlayerActions());
    System.out.println(controller.getActivePlayer().getBalance());
    Assert.assertTrue(1492 == controller.getActivePlayer().getBalance());
}

@Test
public void PayRentWithHouse() throws IOException, InvalidFormatException, NotAProperty {
    GameController controller = new GameController(2);
    controller.getActivePlayer().setLocation(1);
    controller.buyProperty(controller.getBoard().getBoardLocations().get(controller.getActivePlayer().getLocation()));
    controller.getActivePlayer().setLocation(3);
    controller.buyProperty(controller.getBoard().getBoardLocations().get(controller.getActivePlayer().getLocation()));
    controller.buyHouse();
    controller.endTurn();
    controller.getActivePlayer().setLocation(3);
    controller.performActions(controller.getPlayerActions());
    Assert.assertTrue(1480 == controller.getActivePlayer().getBalance());
}
```

Summary of Sprint

This sprint, the frontend team tried to use a new game library to help build the unique frames for auctions and properties. This should have been discussed eaerlier in the development cycle. This new library was later dropped which we feld was a bit of a waste of time. This was the first sprint that we had to hold our sprint meetings remotely as a team member was out of the country. As a team, we coped well and stayed focused.

With the fronted, there was a complete redesign on the user interface using a new service. This has allowed the frontend team to code much easier transitions and connect different scenes together. The disadvantage of this was that the new service required more time on the user interface than should have due to a full redesign.