Real Estate Game

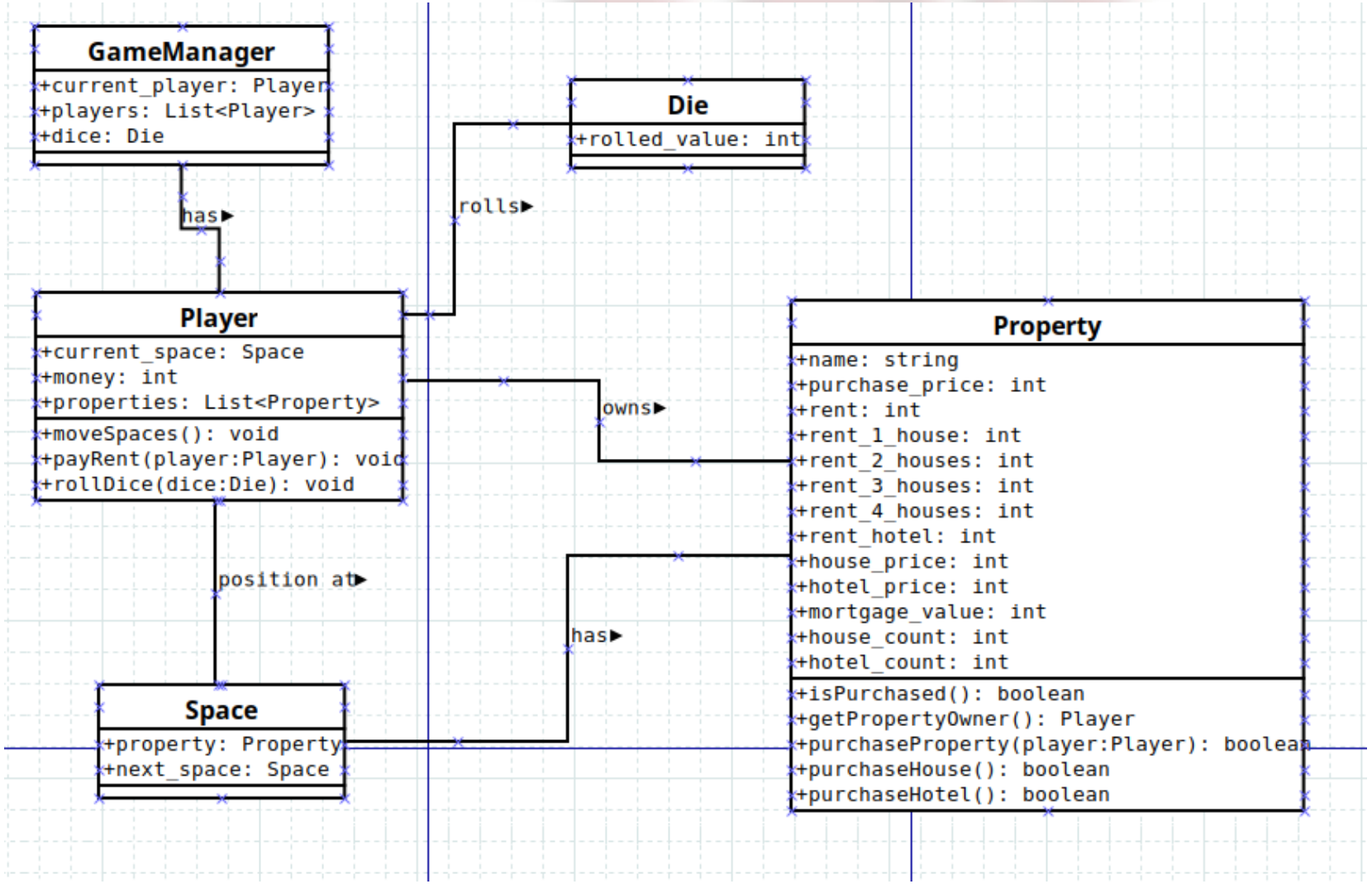
Knox Game Design

December 2025

Overview

- Not for beginner programmers
  - Multiple states and game loops / modes
  - “Build one to throw away” – Fred Brooks, Mythical Man Month
- Create list of rules / requirements
- Get basic gameplay working first. Then work on graphical display and interface.
- Can design/implement in any order. I’m just offering the steps that I used.
- Design will probably be revised during implementation

Software Design – Class Diagram

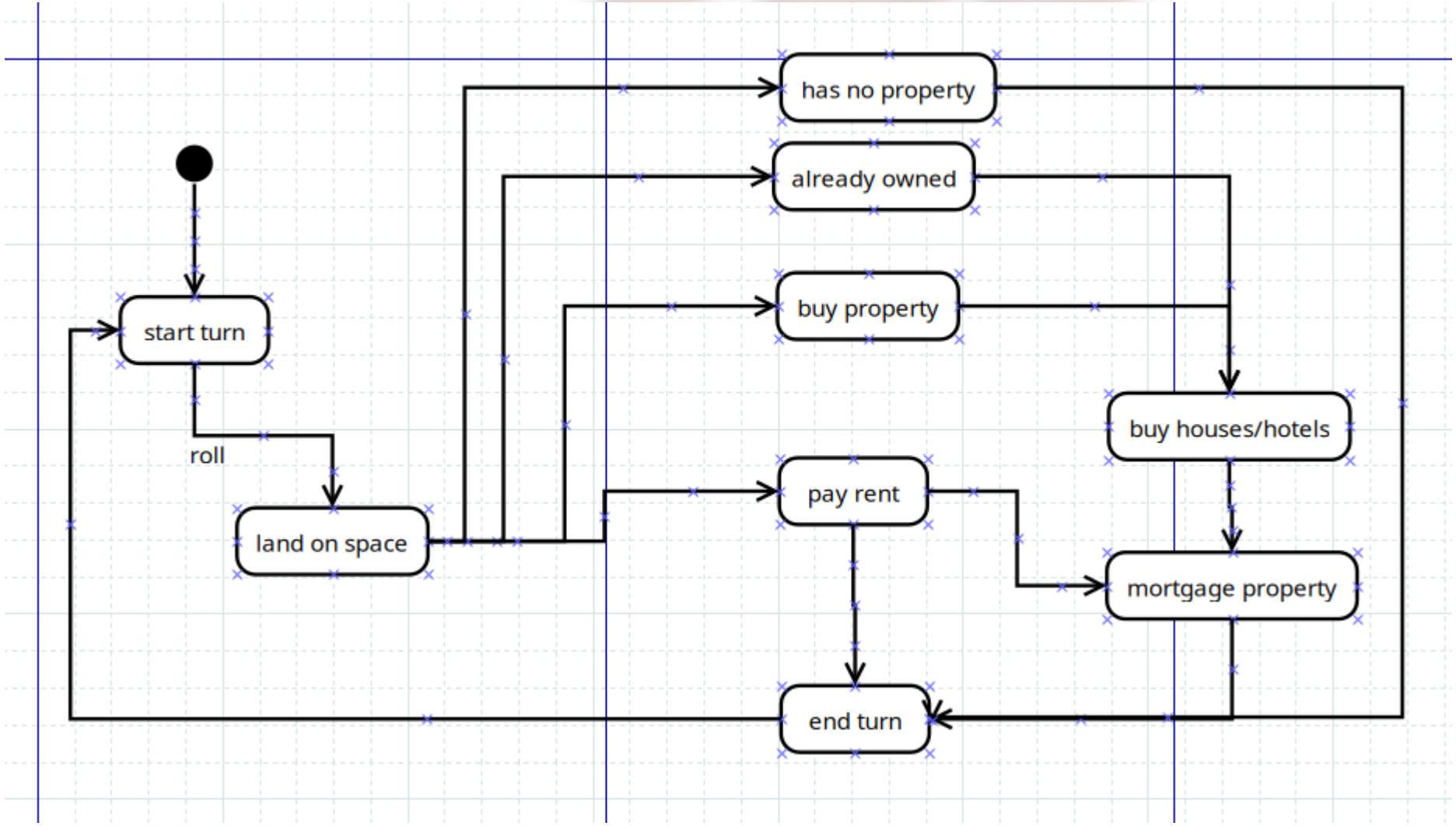


KNOX GAME DESIGN

Design decisions

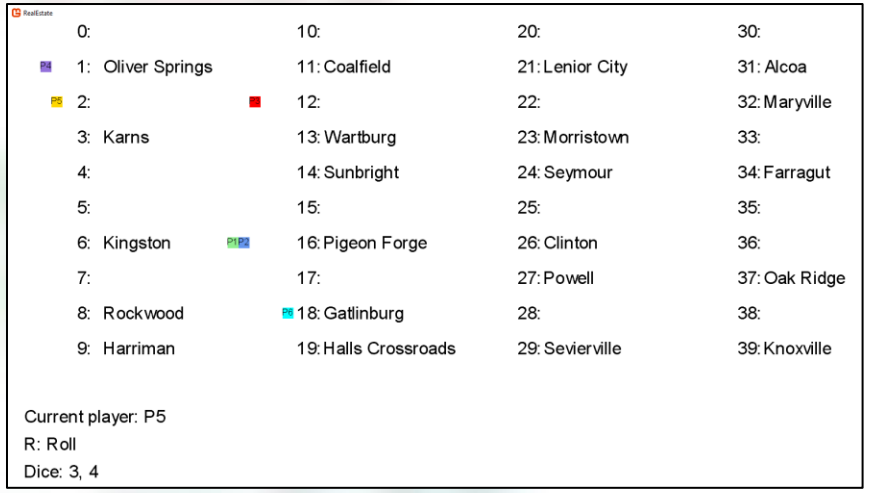
- Linked list of spaces
  - Array would have been more complex
- Purchase methods return boolean (success / failure)
  - Could use exceptions / error handling
- Player has reference to their current space
  - Spaces could have reference to collection of Players on that space
- Player has collection of purchased Property
  - Could have Property have reference to owned Player
  - Issue – multiple players could own same property if not handled correctly
  - Makes looping through Player's owned properties easier
  - Finding the property's owner requires looping through all of the players properties
    - Avoid having player property List AND reference to owner player in Property. Do one or the other, but NOT BOTH.
    - Make getPlayerOwner method in Property to return the property's owner

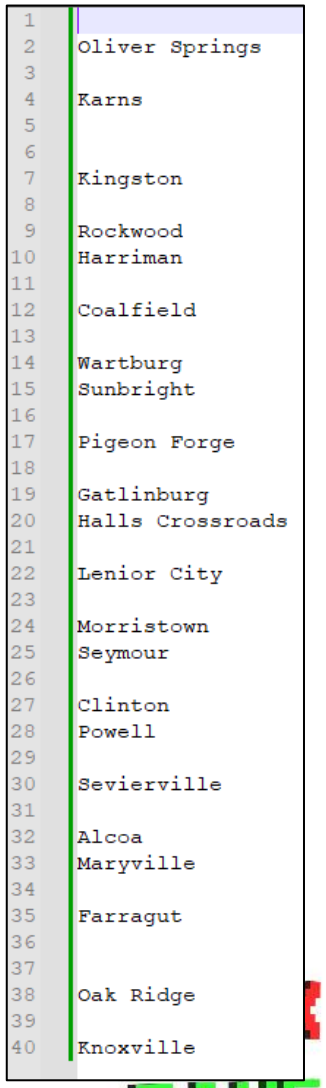
Software Design – State Diagram



Step 1 – Moving players around the board

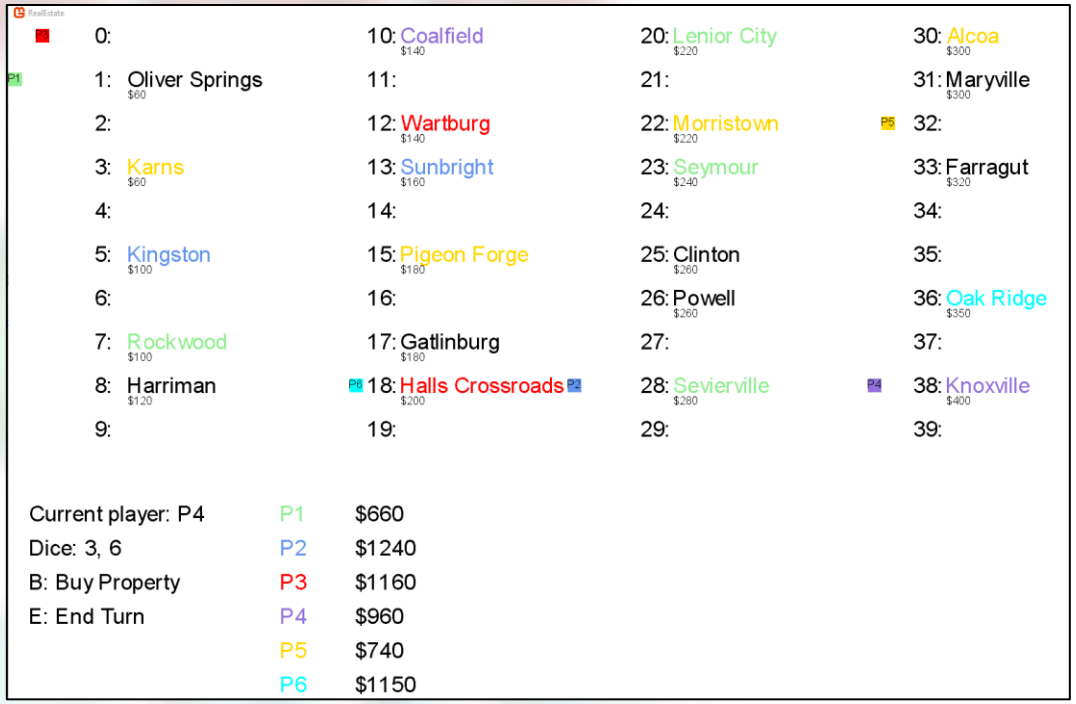
- Load properties list from file
- Create GameManager, Player, Space, Property, and Die classes
- Display objects to screen
- Randomize dice value when ‘R’ pressed
- Move current player number of spaces rolled
- Set current player to next player when turn ends
- Only do enough graphical display to convey current game status





Step 2 – Money / Property Purchase / States

- Give each player an initial amount of game money
  - Design decision – money tracked as integers. not individual bills
- Add cost of each property in data file
  - Design decision – using CSV (comma separated values) format. Could translate to JSON or YAML, but extra parsing code is required.
- Allow player to buy unowned property when stopping on it
  - Player money must be greater than purchase cost
  - Subtract purchase cost from player money
  - Assign instance of property to player
- Add GameState enum
  - StartTurn
  - LandOnSpace
  - EndTurn



Property color is the color of owning Player

```
Oliver Springs,60

Karns,60

Kingston,100

Rockwood,100
Harriman,120

Coalfield,140

Wartburg,140
Sunbright,160

Pigeon Forge,180

Gatlinburg,180
Halls Crossroads,200

Lenior City,220

Morristown,220
Seymour,240

Clinton,260
Powell,260

Sevierville,280

Alcoa,300
Maryville,300

Farragut,320


Oak Ridge,350

Knoxville,400
```

KNOX GAME DESIGN

Step 3 – Rent / Eliminate Player / Game Over

- Add rent values to property data file
- Transfer rent value to property owner when landing on property space
  - Display message stating amount transferred
- Player is eliminated if unable to pay rent
  - Remove player from GameManager players List
  - Win condition – only one player left in players List
    - Transition to GameOver state
  - Test by making rent a really high value (all rents $5000)
    - Remember to set it back after testing
  - When player eliminated, all of their properties become available again, which may not follow "traditional" rules

P5 paid $16 to P6 at Halls Crossroads

P3 unable to pay $5000 at Wartburg. Eliminated from game.

Game Over: Player P3 wins!

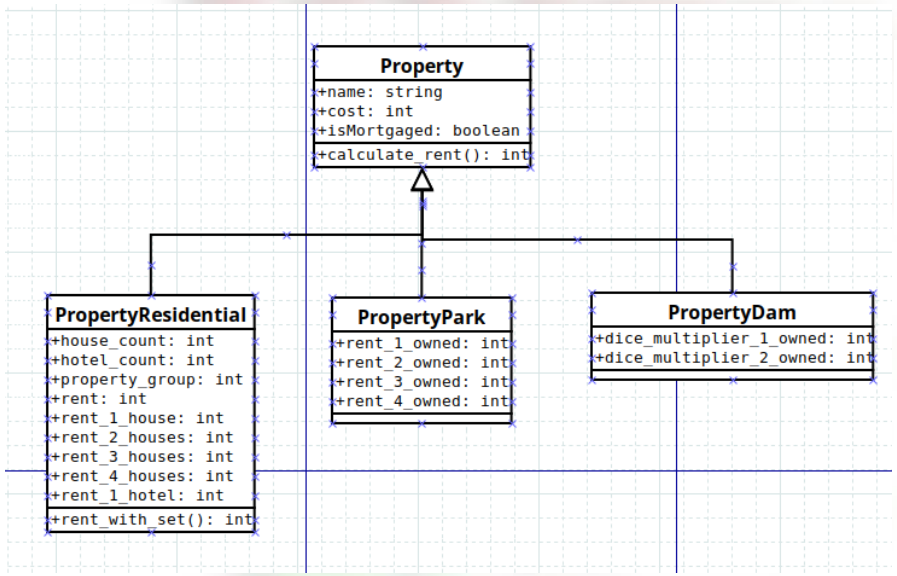
```
1
2   Oliver Springs,60,2
3
4   Karns,60,4
5
6   Kingston,100,6
7
8   Rockwood,100,6
9   Harriman,120,8
10
11  Coalfield,140,10
12
13  Wartburg,140,10
14  Sunbright,160,12
15
16  Pigeon Forge,180,14
17
18  Gatlinburg,180,14
19  Halls Crossroads,200,16
20
21  Lenior City,220,18
22
23  Morristown,220,18
24  Seymour,240,20
25
26  Clinton,260,22
27  Powell,260,22
28
29  Sevierville,280,24
30
31  Alcoa,300,26
32  Maryville,300,26
33
34  Farragut,320,28
35
36
37  Oak Ridge,350,35
38
39  Knoxville,400,50
```

KNOX GAME DESIGN

Step 4 – Ownable Properties / Calculating Rent

- 22 “residential”
  - 8 sets
  - can have 4 houses each
  - can have 1 hotel each
- 4 “parks”
  - rent is based on number owned (1x, 2x, 4x, 8x)
- 2 “dams”
  - rent is based on dice roll and number owned (4x, 10x)

- calculate_rent is abstract method that must be implemented by subclass
- testing requires controlling roll values to land on test case spaces (park/dam). using random rolls would take forever to land on test case spaces
- mortgage_value is actually a calculated value: half the purchase price



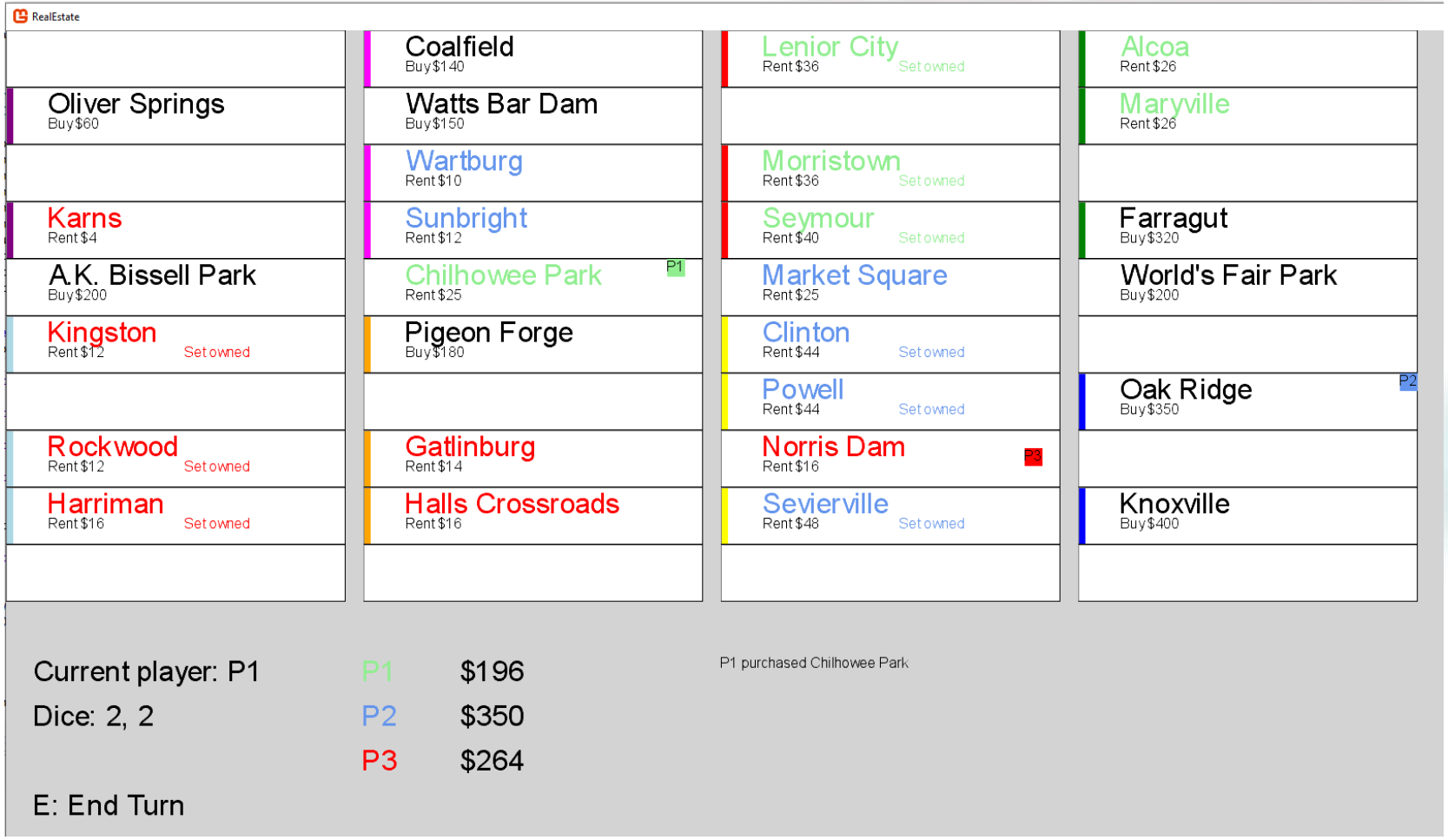
Rent on PropertyPark

```
2 references
public override int calculateRent() {
    int iParkCount;
    iParkCount = getPropertyOwner().getOwnedParkCount();
    return 25 * ((int) MathF.Pow(2, iParkCount - 1));
}
```

Rent on PropertyDam

```
3 references
public override int calculateRent() {
    int iDamCount;
    int iMultiplier;
    iDamCount = getPropertyOwner().getOwnedDamCount();

    switch(iDamCount) {
        case 1:
            iMultiplier = 4;
            break;
        case 2:
            iMultiplier = 10;
            break;
        default:
            return 0;
    }
    return (gamemanager.dice[0].iRolledValue + gamemanager.dice[1].iRolledValue) * iMultiplier;

}
```

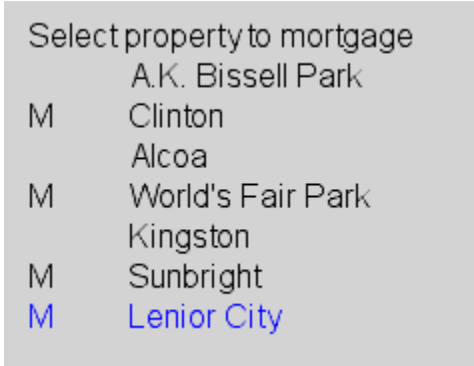
- Add property set id to Residential properties
- Update properties data file with remaining property values
  - Update PropertyResidential calculateRent method to doubled rent if all properties in set owned by the same player
- Update parser to create Residential/Park/Dam based on first CSV token on line




```
R,0,Oliver Springs,60,2,10,30,90,160,250,50,50

R,0,Karns,60,4,20,60,180,320,450,50,50
P,A.K. Bissell Park,200
R,1,Kingston,100,6,30,90,270,400,550,50,50

R,1,Rockwood,100,6,30,90,270,400,550,50,50
R,1,Harriman,120,8,40,100,300,450,600,50,50

R,2,Coalfield,140,10,50,150,450,625,750,100,100
D,Watts Bar Dam,150
R,2,Wartburg,140,10,50,150,450,625,750,100,100
R,2,Sunbright,160,12,60,180,500,700,900,100,100
P,Chilhowee Park,200
R,3,Pigeon Forge,180,14,70,200,550,750,950,100,100

R,3,Gatlinburg,180,14,70,200,550,750,950,100,100
R,3,Halls Crossroads,200,16,80,220,600,800,1000,100,100

R,4,Lenior City,220,18,90,250,700,875,1050,150,150

R,4,Morristown,220,18,90,250,700,875,1050,150,150
R,4,Seymour,240,20,100,300,750,925,1100,150,150
P,Market Square,200
R,5,Clinton,260,22,110,330,800,975,1150,150,150
R,5,Powell,260,22,110,330,800,975,1150,150,150
D,Norris Dam,150
R,5,Sevierville,280,24,120,360,850,1025,1200,150,150

R,6,Alcoa,300,26,130,390,900,1100,1275,200,200
R,6,Maryville,300,26,130,390,900,1100,1275,200,200

R,6,Farragut,320,28,150,450,1000,1200,1400,200,200
P,World's Fair Park,200

R,7,Oak Ridge,350,35,175,500,1100,1300,1500,200,200

R,7,Knoxville,400,50,200,600,1400,1700,2000,200,200
```

Step 5 – Mortgage / Unmortgage

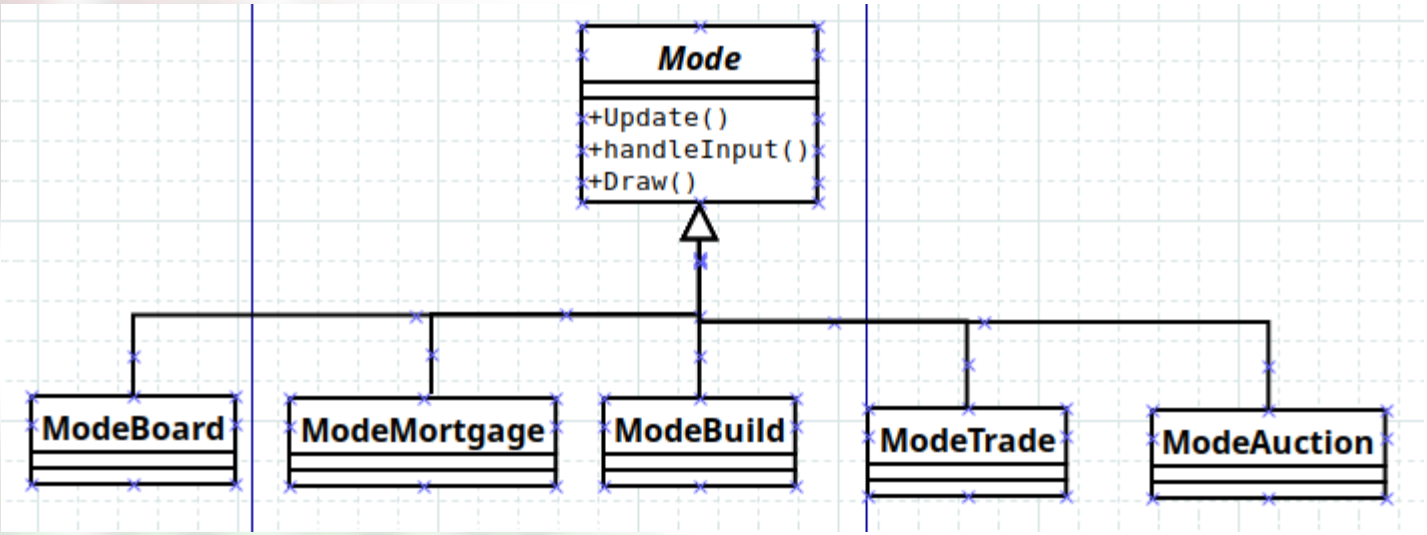
- Need a separate game loop for selecting properties to mortgage / unmortgage
- Mortgage value is half the purchase cost – use method instead of storing in variable
- Update rent collection code to not charge when landing on mortgaged property
- Allow unmortgage if players money is greater than or equal to mortgage value * 1.1

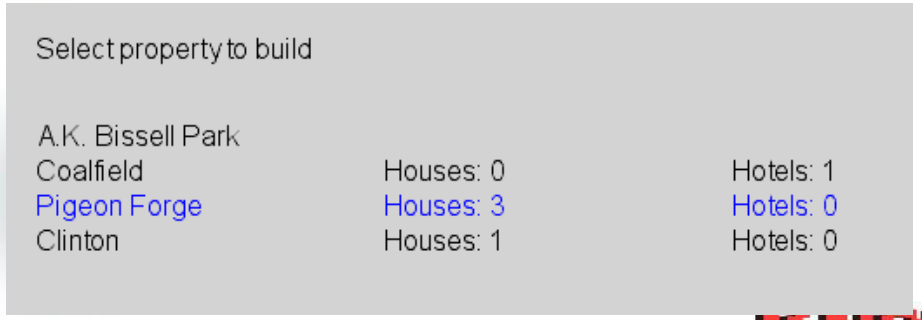


Clinton is mortgaged. No rent paid

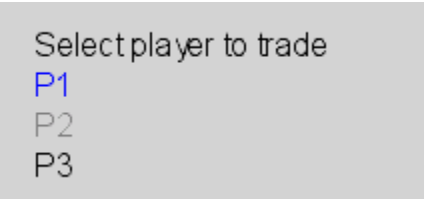
Step 6 - Game Modes / Building Houses and Hotels

- Modes to separate distinct functions
- Track current mode in the GameManager
- Modes implement Update, and Draw methods
- Build mode
  - Allow player to buy houses and hotels if they own all properties in set
  - Remember to recalculate rent value based on house / hotel count
  - Must check that property class type is PropertyResidential, then cast it to use the house / hotel properties

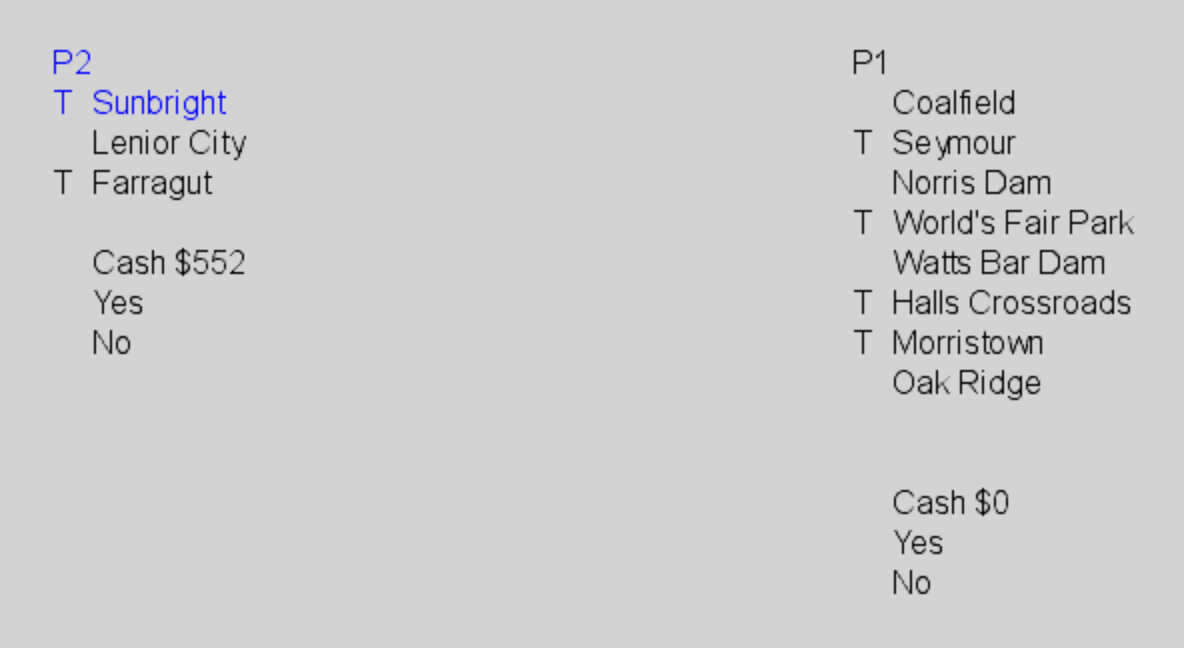




Step 7 - Trade

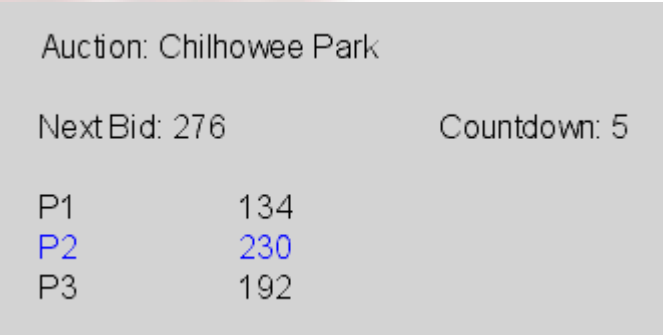


- Select player to trade with
  - Build list of properties to trade for each player
  - Allow specification of cash amount
  - Add Yes / No option for both traders
  - Return to trading loop to make changes if one player chooses No
  - Resolve when both players select Yes or trade is cancelled
  - Object oriented makes moving properties between players easy
    - Just move the reference (pointer) from one player to the other
- Trade screen is a “project in itself”
  - Having modes allows development on trade screen separate from the rest of the game

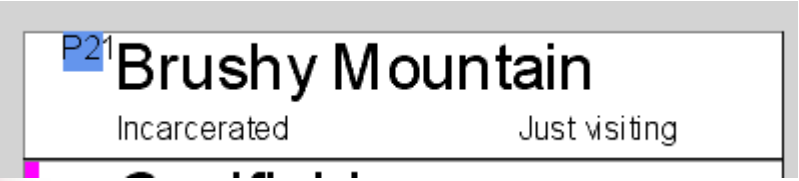


Step 8 - Auction

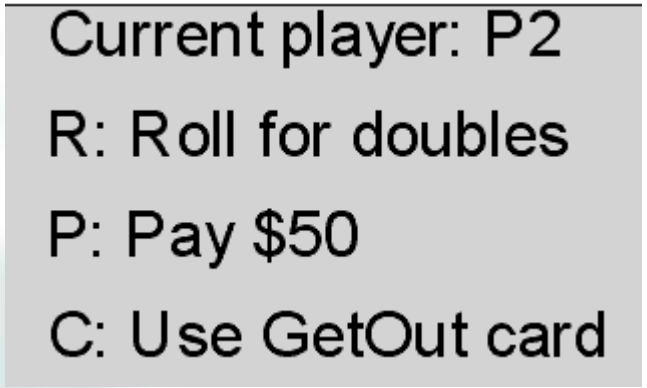
- Allow property to be auctioned when player lands on property space
- Auction is its own mode and interface
- Set countdown timer for auction to complete
  - Reset countdown after a player bids
- Assign property to player with highest bid
- Design decisions
  - 10 seconds to close after last bid
  - Starting bid is 0.10 * purchase price
  - Next bid is 1.20 * previous bid
  - These values can be modified as needed



Step 9 – Doubles, Arrested, Incarceration



- Allow player to roll again on doubles if not incarcerated
- Make Incarceration space a subclass of Space class
  - Incarceration space has a List/collection of incarcerated players
- Move player to Incarceration space when landing on the Arrested space
- Track count of times player has rolled doubles in a row
  - Send to incarceration if doubles rolled three times
- Incarcerated options
  - Roll for doubles: die1.value == die2.value
  - Pay: currentPlayer.iMoney -= 50
  - Card: currentPlayer.iGetOutCard--
    - Add instance variable to Player to track GetOut cards
- Track whether player is incarcerated or “just visiting”
- Incarcerated loop is separate branch from move spaces loop
- Track number of incarcerated rolls

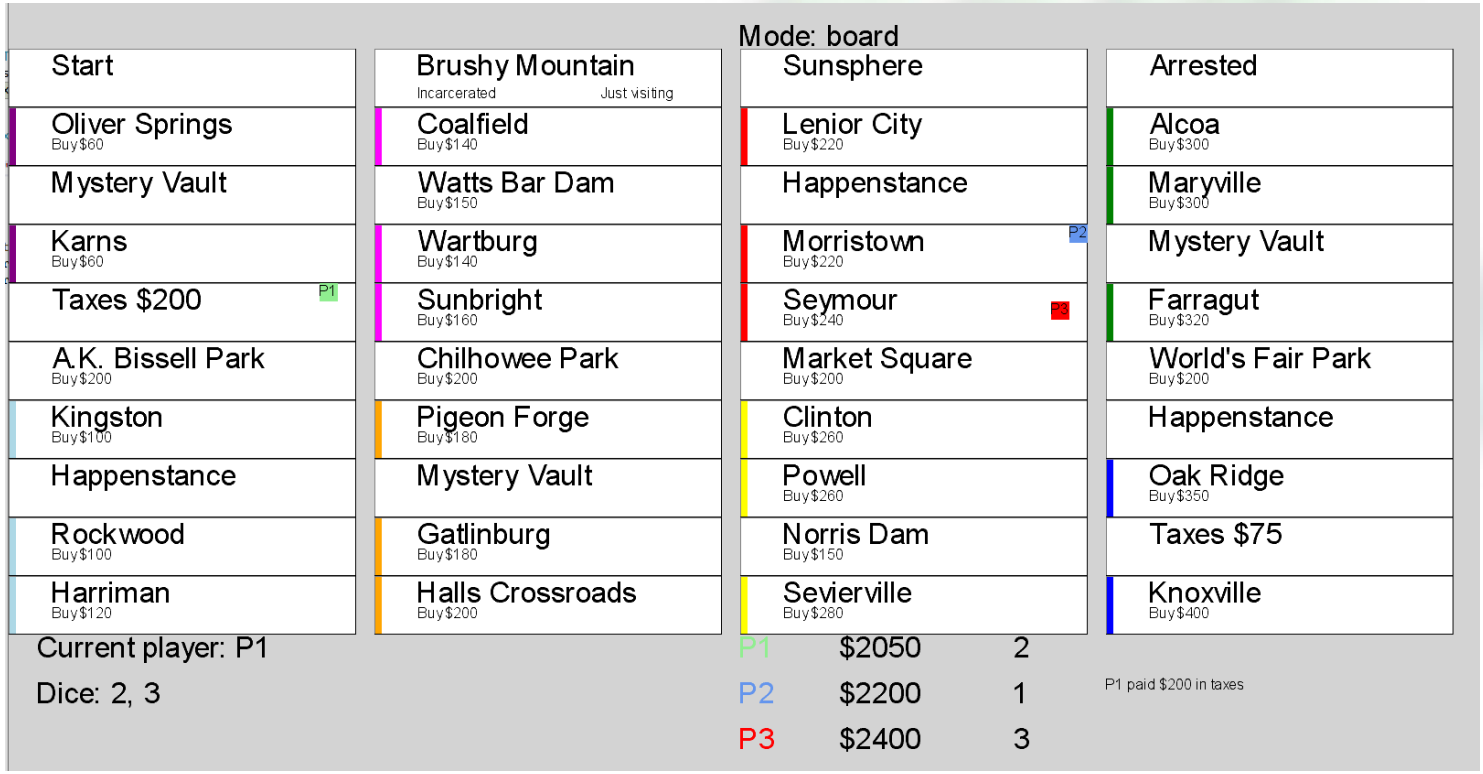


Step 10 – Event Cards

- Event card space a subclass of space
- New EventCard class
- For each card, implement an EventCard subclass
- Add cards to two card lists based on event type
- Randomize the order of the two Lists of event cards
- Draw an EventCard when landing on an EventCard space based on EventCard type
- Implement abstract *action* method on the EventCard class to perform card's instruction in game
  - Good example of using polymorphism in a game
- Only implemented three cards of each type for this demo

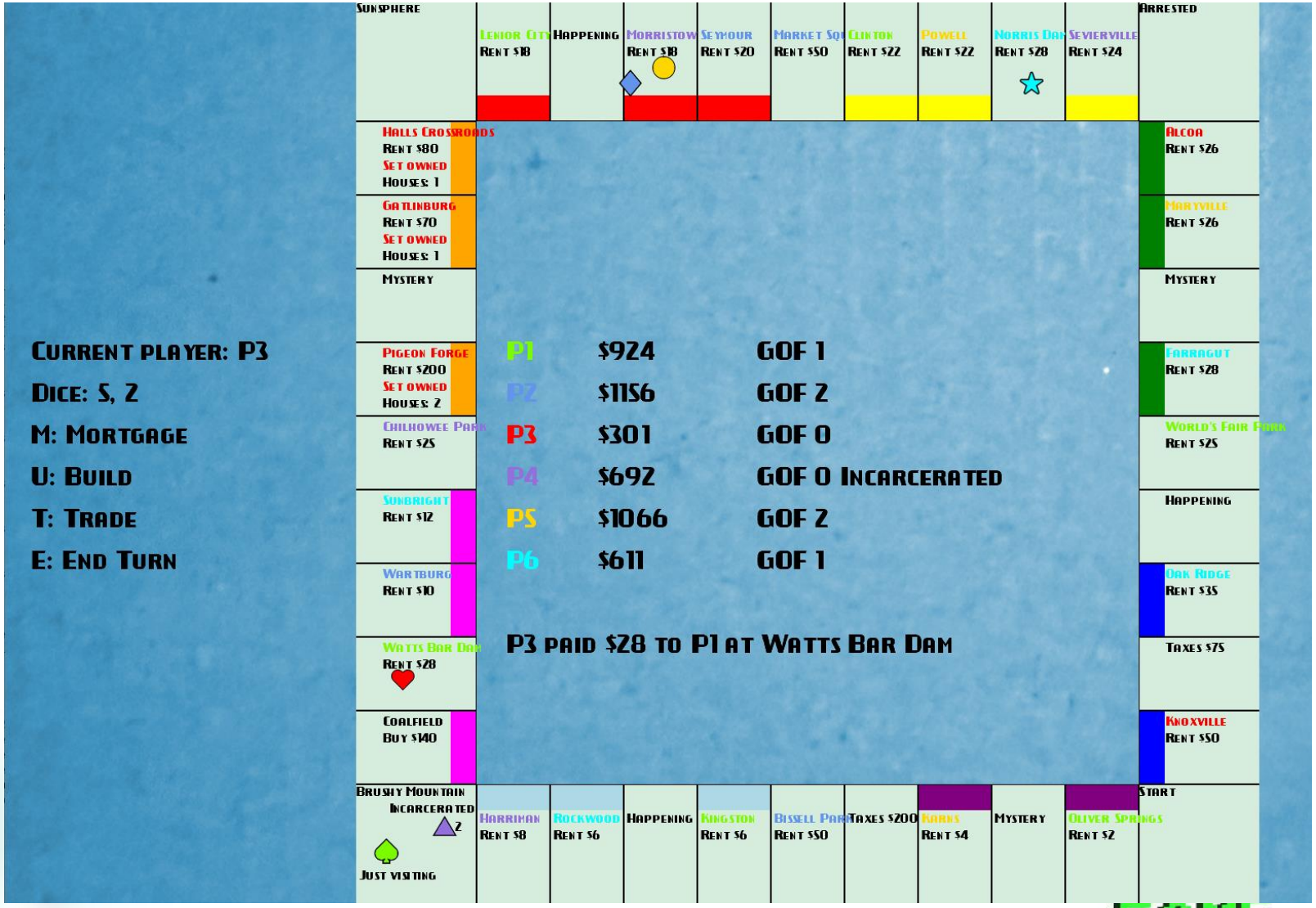
Step 11 – Other spaces

- Start – Earn $200 when landing or passing
- Sunsphere – Free place to park. No action for this space
- Taxes – Pay the required tax amount



Step 12 – Display Manager

- Update graphics / model view controller
  - Use Display manager and display objects for drawing
  - Model game objects should not have drawing methods
  - Separate the “model” from the “view”
  - Display manager has reference to game manager with references to all game objects
  - Allows switching display presentations (text, 2D, 3D, etc)
- Set target position for moving players
  - Start by just using a linear interpolation from current space to next space
    - This is “good enough” for this demo
  - Add DisplayPlayerToken class to separate token display from player model data
  - Current Player model position will be “ahead” of the token display position. Token display will “catch up” with the model
  - To avoid cutting corners when moving, a list of spaces moved will need to be tracked



GAME DESIGN

What else?

- Finish event cards
- Add sound effects and music
- Allow player to sell / mortgage properties to pay debt (another loop)
- Require player to build houses / hotels evenly between property set