- Changes to the state of the game are implemented too generally — there's no way to determine which changes in cash, for instance, pay for what (houses, rents, mortgages, etc.)
- Multiple changes often must happen together in order for them to be legal

↓

- Redefine the GameStateChange and GroupOfChanges objects as follows:

Defs
  I) The Rules of the Monopoly game define what can legally happen in a game and how the game should proceed
  II) A GameState at any instant represents a legal state of the game — one that can be obtained by following the Rules

We have previously listed all the possible changes to the GameState:

1) Player changes:
   a. Cash
   b. Position
   c. Properties to add
   d. Properties to remove
   e. Num. "in-jail" moves
   f. Num. "Get out of jail free" cards
   g. "In-game" status
2) Property changes:
   a. Num. houses/hotels
   b. "Mortgaged" status

However, many of these changes are inherently intertwined, requiring (by the Rules) that other changes be made to maintain Def II (i.e. a legal state).

Ex
  Buying a property requires both changes 1a and 1c

It should never be possible for a GameStateChange to implement one such change without the other(s).

So, we arrive at two new definitions:

III) A GameStateChange represents the smallest set of changes that must be made to transition from one legal GameState to another legal GameState.

(By "smallest", I mean the least amount of changes required to be legal. Using the example above, this could be a single GameStateChange. However, additionally building a house on the newly purchased property in the same GameStateChange would be legal, but would not be the "smallest" possible change. In this case, we would break this into 2 GameStateChanges – one for purchasing the property, and one for building the house.)

IV) A GroupOfChanges is a set of multiple GameStateChanges that, legally, can happen independently, but, perhaps due to a Player's preferences, must be applied together as one unit

(This is useful when, for instance, Players submit building requests. One Player may mortgage a property to get the money to build houses, which involves two GameStateChanges, but would only wish to mortgage the property if he will also get to build the house. These would be grouped in a GroupOfChanges so they can be treated as one unit to reflect the Player's preference.)

Anywhere we would normally return a single GameStateChange, we will instead return a GroupOfChanges, which will then be applied* as one unit. This will guarantee that legality of the GameState is maintained.

*Note: This means that GameStateChanges are no longer applied directly. They are applied indirectly through a GroupOfChanges

"Anywhere", that is, except for the notification process. This will be treated differently because the building changes each Player submits must be resolved and approved before being blindly applied. During this process, some changes may never be applied, and some new ones may even be created (as we'll see later). How this will be handled will be discussed later.

Below is a list of all possible legal transitions between two GameStates. Each of these will be implemented as a static, indivisible Game State Change:

1) Transfer money (player_from, player_to, amount) # One of these players can be the Bank
2) Change position (player, new_position)

Properties

3) Buy property (player, property, mortgaged = False)
4) Transfer property (player_from, player_to, property)
5) Mortgage (property, bank)
6) Unmortgage (property, bank)

Building

7) Build house (property, bank)
8) Build hotel (property, bank)
9) Demolish house (property, bank)
10) Demolish hotel (property, bank)

Jail

11) Send to jail (player)
12) Decrement "in jail" moves (player)
13) Leave jail (player)
14) Increment "Get Out of Jail Free" count (player)
15) Decrement ________________ (player)

16) Eliminate (player_eliminated, player_eliminator)

Now, we will address how these new conceptions of GameStateChange and GroupOfChanges are used in the notification process.

When multiple Players want to build at the same time, they may run into conflicts – a housing shortage, a hotel demolition seeking to steal the last 4 houses before someone builds one, etc. After much delibration, we have arrived at a set of rules that settle these conflicts, and they are in the "design" directory. (They are handled by a HousingResolver.)

However, these rules may require that some Players' building changes get rejected (i.e. never applied) in the event of an auction, for instance. They also require that we have access to specific information about the Players' building changes: how many houses is Player x building, demolishing, etc.?

So, in response to a notification, a Player cannot simply return a GroupOfChanges (which can't really contain more than 1 house built/demolished), as many changes may be requested (many houses may be built/demolished). Furthermore, a Player can't return a list of GroupOfChanges objects, as the HousingResolver needs specific information about some changes while ignoring others. Lastly, we need to separate out building changes from any other changes (such as the results of a trade), as we have in the past.

Thus, we arrive at two more definitions:

V) A BuildingRequests object represents a set of building changes requested by a Player. It contains:

- 4 GroupOfChanges objects, one for each:
  1) Houses Built
  2) Houses Demolished
  3) Hotels Built
  4) Hotels Demolished

VI) A NotificationChanges object is a container for all changes/requests made by a Player in response to a notification. (It is analogous to our current use of the GroupOfChanges — before I wrote this document.) It contains:

- A list of GroupOfChanges objects for all non-building-related changes (one GroupOfChanges for a trade with Player x, another for a trade with Player y, for instance)
- A BuildingRequests object for all building requests to be considered separately

In response to a notification, a Player will return a NotificationChanges object. The list of non-building-related GroupOfChanges objects will be applied immediately. Then, the HousingResolver will receive a dictionary mapping Players to their BuildingRequests, work out 'who builds what', and apply those changes directly to the GameState*.

* see the explanation of how the HousingResolver works to see why it applies changes directly rather than returning a GroupOfChanges.

This concludes the notification process.

Below are diagrams of the new objects defined in this document

III - GameStateChange

Instance Vars

- All the same attributes as before

Methods

- One static method to return each type of GameStateChange listed on page 3

IV - Group Of Changes

Instance Vars

- Game State Change [ ]

Methods

- Getter for GameStateChange [ ]

V - Building Requests

Instance Vars

- Houses Built (GroupOfChanges)
- House Demo'd (GroupOfChanges)
- Hotels Built (GroupOfChanges)
- Hotels Demo'd (GroupOfChanges)

Methods

- Getters for each instance var. (GroupOfChanges)
- Getters for each instance var. as a quantity of houses built, demo'd, etc.

VI - Notification Changes

Instance Vars

- Non-Building Changes (GroupOfChanges[ ])
- Building Requests (BuildingRequests)

Methods

- Getters for each instance var.