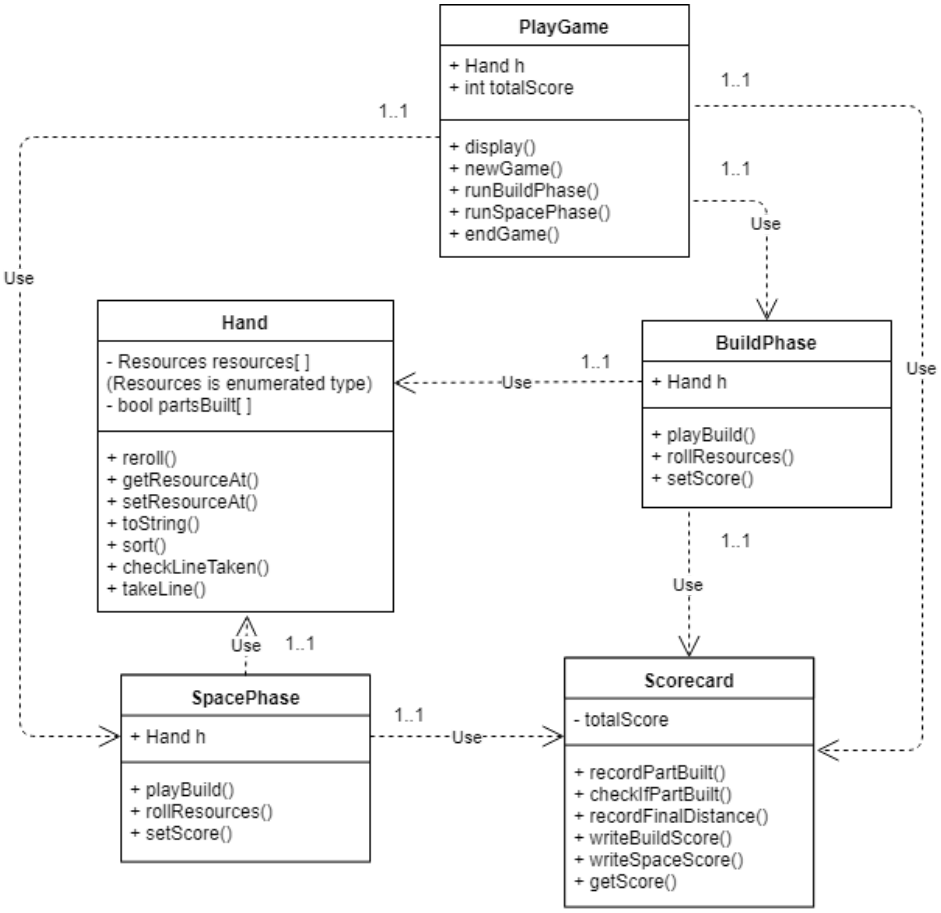
Kevin Hance, Andrew Yang, and Zach McKee
CPSC 224-01 (Professor Zhang)
4/10/18
Design Alternative Analysis

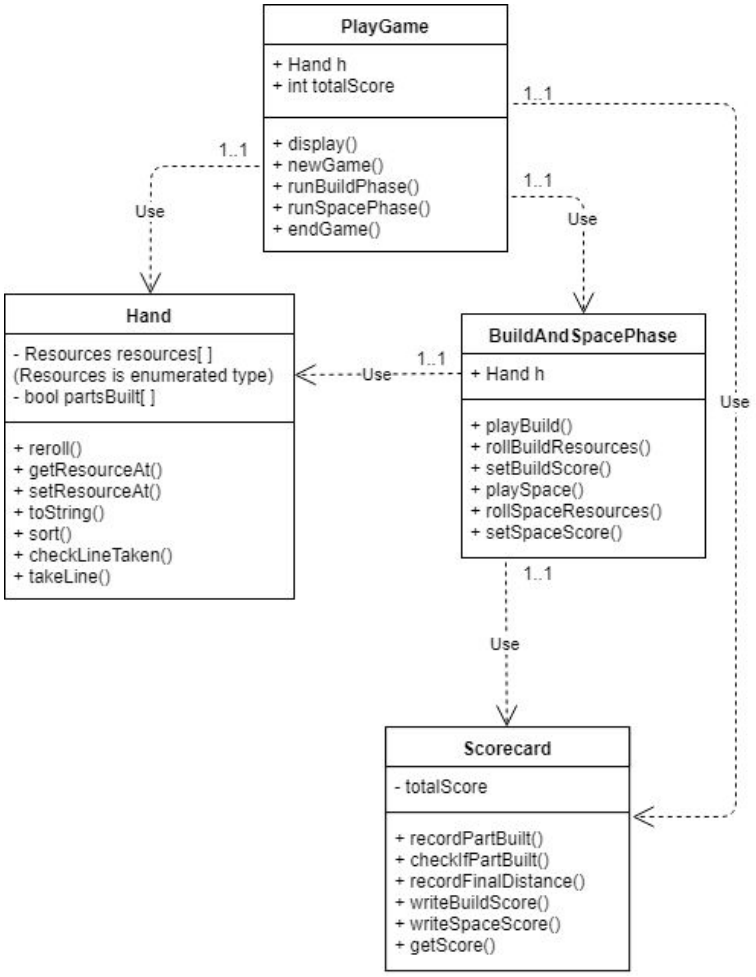
1. After designing the UML for the Preliminary Class Design, we continued to work on our code project documentation and source code. It quickly became apparent that our Preliminary Class Design was a very skeletal outline of what our program would turn out to look like. Our first alternative design is an updated version of our Preliminary Design Analysis that incorporates our new additions.



Pros of this design: PlayGame, BuildPhase, and SpacePhase can all keep track of the Hand. Scorecard calculates and keeps track of the total score, PlayGame stores result. Separate functions to execute BuildPhase and SpacePhase.

Cons of this design: Separating BuildPhase and SpacePhase into two separate classes may result in code redundancy. Scorecard needs to be able to calculate scores for both BuildPhase and SpacePhase.

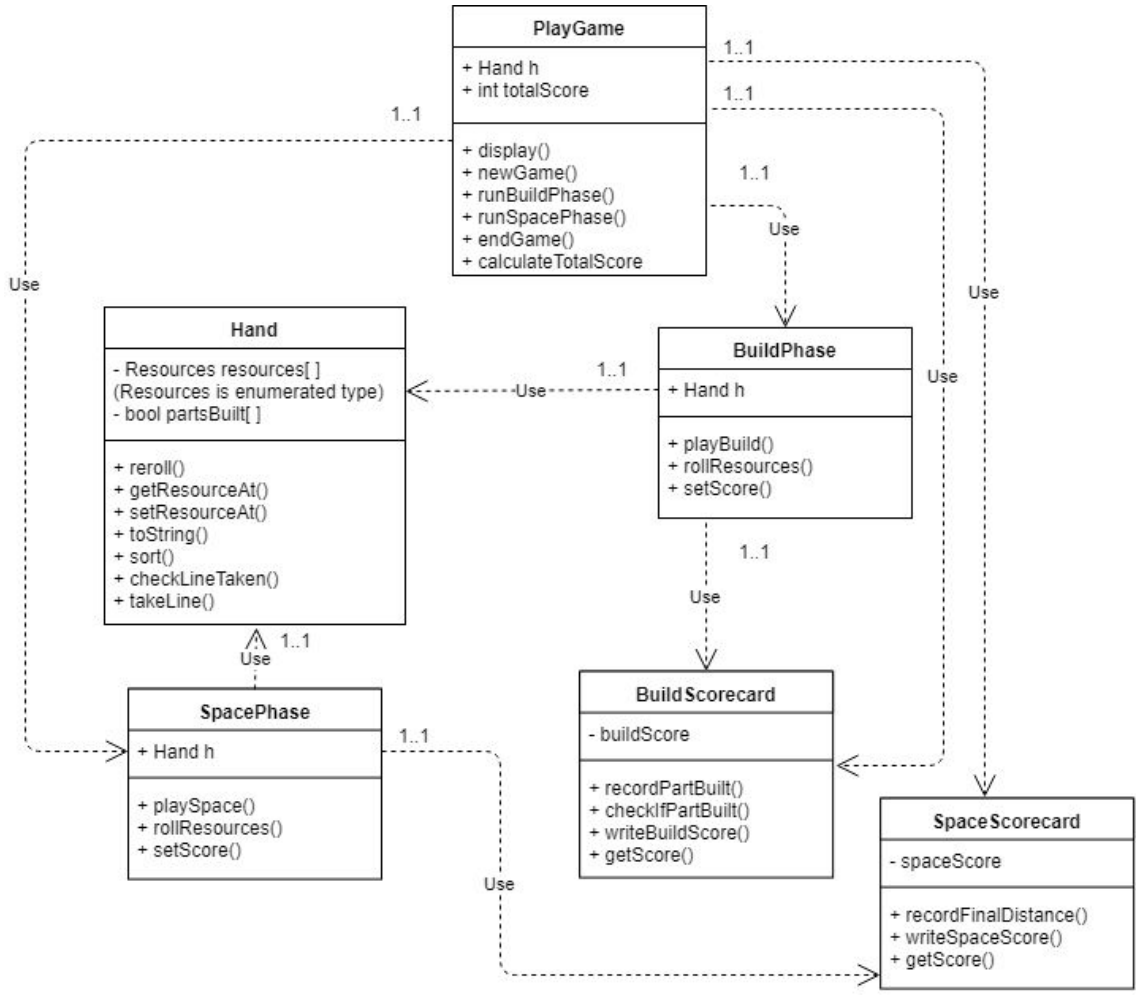
2. While designing the layout for the various classes in our project, it became apparent that having two separate classes for the two separate game phases may be problematic. Specifically, we divided the game up into two distinct parts, the “Build Phase” and the “Space Phase”. While both use similar rules and dice-rolling implementation, the player has completely different goals in each phase. To counteract any possible issues from separating these into two distinct classes, we designed an alternative UML layout that consolidated both phases into one class.



Pros of this design: BuildPhase and SpacePhase are consolidated into one class. Therefore, there will ideally be much less redundancy in the class design.

Cons of this design: Putting BuildPhase and SpacePhase into one class may result in a class that looks “crowded” and it hard to read. To counteract this, we must document our code well with comments to avoid confusion. Scorecard still needs to be able to calculate scores for both BuildPhase and SpacePhase.

3. While designing the game rules for this project, it became apparent that the two different phases of our game (Build Phase and Space Phase) are evaluated in completely different ways when the score is calculated for either of the two. Because of this, we decided to design an alternative UML layout that features two more scoreboard classes (BuildScoreboard and SpaceScoreboard) to split up the responsibilities of calculating score. The total score is then calculated using a public function in the PlayGame class.



Pros of this design: Separate functions to execute BuildPhase and SpacePhase. Separate Scorecard classes for the BuildPhase and the SpacePhase, as the two have their respective scores calculated in completely different ways.

Cons of this design: Creating two separate Scorecards for BuildPhase and SpacePhase may result in some redundant code. In addition, this method would take up more memory, as data structures used to keep track of the players score would need to exist in both classes.