**Phase 10 - Design Document**

Max Robinson, Justice Nichols, Danny Nelson, Hunter Torset

31 October 2014

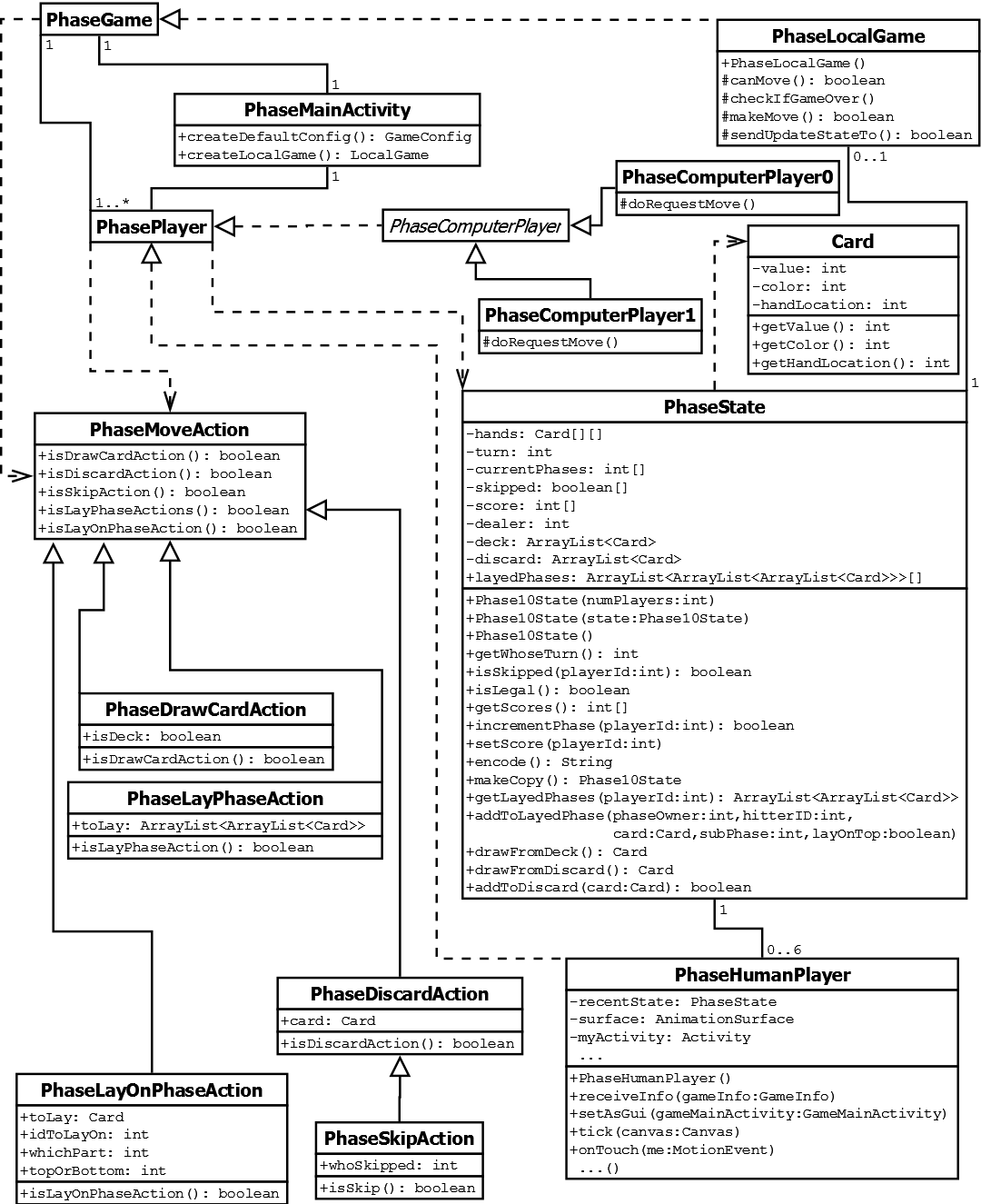
This document describes the design of the computerized Phase 10 game. The goal is to produce a game that is consistent with the requirements described in “Phase 10 - Requirements Document”, dated 3 October 2014.

**Overview of the class structure**

The game will consist of the following classes and interfaces (in addition to those supplied by the game framework)

* Phase 10 game
  + PhaseGame - interface that defines a Phase10 Game
  + PhaseLocalGame - object that interacts with players and enforces all the rules
* Phase 10 - player
  + PhasePlayer - interface that defines a Phase 10 player
  + PhaseComputerPlayer - abstract class that defines a computerized phase 10 player
  + PhaseComputerPlayer0 - player that makes random moves
  + PhaseComputerPlayer1 - player that plays with some semblance of intelligence
  + PhaseHumanPlayer - GUI-based player that allows a human user to play
* Phase 10 State
  + PhaseState - contains the entire state of the Phase 10 table top
* Phase 10 move-actions
  + PhaseMoveAction - abstract class that defines a Phase 10 move action
  + PhaseDrawCardAction - draw a card from the deck
  + PhaseDiscardAction - discard a card from hand to the discard pile
  + PhaseSkipAction - discard a skip card from hand and choose a player to skip
  + PhaseLayPhaseAction - lay a phase down from hand
  + PhaseLayOnPhaseAction - play a card on an already laid phase
* Driver
  + PhaseMainActivity - class that defines the startup screen

Figure 1 shows the UML class diagram that illustrates the relationships between the classes. To avoid cluttering the diagram, game framework classes are not shown; the names of the classes however suggest the inheritance connection.

**Figure 1: UML Class Diagram**

Consistent with game framework, player objects will never interact. Rather each player will interact with the game object; when necessary, the game object will forward information onto the other player.

**Individual Class Discussion**

PhaseState

Phase state objects are to be used for two different purposes:

* Sent by the game to a player to inform of a change in the game’s state. (This is the traditional use for a GameState object in the framework).
* Used internally by the PhaseLocalGame object to maintain the “official” state of the game.
  + To maintain the integrity of the game’s state, the PhaseLocalState object is never seen by the players. Whenever a player requests the game’s state, it is given a copy of the PhaseLocalGame’s PhaseState object.

Phase 10 is not a perfect-information game, thus there is a need to hide parts of the game state’s information. After the copy of the PhaseLocalGame’s PhaseState object has been made, the other player’s hands as well as the deck must be nulled out before the PhaseState object is able to be sent to the player.

Within the PhaseState object, the following information is kept:

* The cards each player has in their hand.
  + private Card[][] hands;
* The player’s ID whose turn it currently is.
  + private int turn;
* Phase each player is currently trying to complete.
  + private int[] curentPhases
* Players who will be skipped the next time it is their turn. True means the player will be skipped on their next turn, false means the player will be able to play on their next turn.
  + private boolean[] skipped;
* While dealing of the cards will be handled internally and invisibly to the players, it is still important to store the current position that the dealer is in to denote who will have the first turn. Positions will be labeled 1 - 6.
  + private int dealer;
* Cards that are currently on the deck will be stored in the order there they will appear. The ArrayList remove method will be called on the ArrayList to remove the 0th position card if the deck contains one or more cards. In the case where the deck does not contain any cards, all but the top card in the discard pile will be removed and shuffled. The result of this operation will become the new deck.
  + private ArrayList<Card> Deck;
* Cards that have been discarded from player’s hands will be stored in the discard pile. Players will be able to remove element 0 of the discard pile, and must replace the card with one from their hand. This will be accomplished by using the ArrayLists insert and remove methods.
  + private ArrayList<Card> discard;
* Phases that players have laid down will be stored. This variable will contain each players laid phase, which if the current phase requires it, can be divided into two portions.
  + private ArrayList<ArrayList<ArrayList<Card>>>[]

The following PhaseState methods are available. Details on these methods may be found in the accompanying javadoc-style documentation. It is expected that the following methods - which do not change the PhaseLocalGame’s PhaseState object’s state - will be used by both games and players:

* **int getWhoseTurn() -** whose turn it is, corresponds to the player ID.
* **boolean isSkipped(int playerID) -** returns if the player whose ID is passed into the function will be skipped on their next turn.
* **int getScores() -** get the current scores of all of the players
* **String encode() -** encode the PhaseState object into a string, that when passed as a parameter to the constructor, creates an identical copy of the state. This string will consist of integer values seperated by commas, denoting the value of variables in the order listed above. The are a total of **CHECK,** values
* **ArrayList<ArrayList<Card>> getLayedPhase(int playerID) -** the current cards in the player whose ID is specified.

The following methods modify the state:

* **boolean incrementPhase(int playerID) -** takes a player to the next phase.
* **void setScore(int playerID, int score) -** sets the player whose ID is specified to the score.
* **addToLayedPhase(int phaseOwner, int hitterID, Card card, int subPhase, int topOrBottom)-** allows players to hit on another player’s laid phases. The phaseOwner is the id of the player whose phase is being played on. Card is the card that is being played. SubPhase is which part of the players phase is being played on with values 0 or 1. TopOrButtom says which end of a subPhase a card is being played on with values 0 or 1.
* **Card drawFromDeck() -** removes the top card (the card in position 0) from the deck if there is one or more cards on the deck. In the event that there are no cards left on the deck, all but the first card in the discard pile will be shuffled and place back into the deck. After this has completed, the cards will then be drawn.
* **Card drawFromDiscard() -** removes the top card (the card in position 0) from the discard pile.
* **void addToDiscard(Card card) -** the specified card will be added to the to the top (position 0) of the discard pile.

PhaseMoveAction and subclasses

PhaseMoveAction is an abstract class that represents a generic action by a player. There are five subclasses:

* PhaseDrawAction - Draw a card from the deck
* PhaseDiscardAction - Discard a card and place into discard pile
* PhaseSkipAction - Discard a skip card
* PhaseLayPhaseAction - Lay down a completed phase
* PhaseLayOnPhaseAction - Lay down a card out of hand on a currently laid phase

The PhaseMoveAction class defines a predicate for each of these classes that tells whether a given PhaseMoveAction belongs to the class. These are:

* **boolean isDrawAction()** - is this object a **PhaseDrawAction?**
* **boolean isDiscardAction()** - is this object a **PhaseDiscardAction?**
* **boolean isSkipAction()** - is this object a **PhaseSkipAction?**
* **boolean isLayPhaseAction()** - is this object a **PhaseLayPhaseAction?**
* **boolean isLayOnPhaseAction()** - is this object a **PhaseLayOnPhaseAction?**

Instances of PhaseDrawAction contains a boolean that indicates which pile is drawn from. Thus the constructor takes one additional parameter containing a boolean indicating if the draw pile was drawn from.

* **public PhaseDrawAction(GamePlayer p, Card card)**
* **@override public boolean isDiscardAction() { return true; }**

Instances of PhaseDiscardAction contains the card that is currently being discarded. Thus the constructor takes one additional parameter of the card to be discarded:

* **public PhaseDiscardAction(GamePlayer p, Card card)**
* **@override public boolean isDiscardAction() { return true; }**

Instances of PhaseSkipAction will contain the ID of the player that is going to be skipped on their next turn. This the constructor takes an additional parameter containing the player to apply the skip to:

* **public PhaseSkipAction(GamePlayer p, int whoSkipped, Card card)**
* **@override public boolean isSkipAction() { return true; }**

Instances of the PhaseLayPhaseAction class contain the phase sets in the correct order that will be laid. The constructor takes an additional parameter containing the sets to be laid as the phase.

* **public PhaseLayPhaseAction(GamePlayer p, ArrayList<ArrayList<Card>> Phase)**
* **@override public boolean isLayPhaseAction() { return true; }**

The final action class is the PhaseLayOnPhaseAction action. This will contain the card to be laid, the ID of the player to lay this card on, the set of the phase to lay card on, and whether to put the card on the top or the bottom of the currently laid phase (true - top, false - bottom). The constructor is parameterized as follow to handle this action:

* **public PhaseLayOnPhaseAction(GamePlayer p, Card toLay, int idToLayOn, int whichPart, boolean topOrBottom)**
* **@override public boolean isLayOnPhaseAction() { return true; }**

PhaseGame and PhasePlayer

PhaseGamePlayer and PhasePlayer are “placeholder” interfaces that do not define any additional abstract methods beyond the Game and Player interfaces that they extend.

PhaseLocalGame

PhaseLocalGame is one of the most crucial and integral components of the game as it performs many duties. This class maintains the game’s official state, communicates with players, and enforces the rules of the game.

It maintains the PhaseState object as the game’s official state. It prevents players from modifying this state by keeping the official copy hidden - when a player asks for the game state, this player will be sent a copy of the official game state. In this copy, the deck and the other players hands will be nulled out as Phase 10 is not a perfect information game.

When the PhaseLocalGame object receives a move from a player (makeMove method), it first check to see if it was that players turn then check to see what kind of move is being made, and acts accordingly:

* if its a draw action, a check is made that it is the players turn. If the players hand contains 10 or less cards, the player’s hand in PhaseState is then changed to include the top card off the deck (position 0).
* if a player draws a card from the deck or discard pile, the player’s hand will be checked to see if they have 10 or less cards. If they do, their hand in the PhaseState object will be updated to contain this card that was drawn off the top of the deck.
* if a card is placed on the discard pile, the card will then be checked to determine if it is a skip card. If it was, this will trigger a message to the player to select who they would like to skip and a skipAction will be created. If it is any other card, the game state is changed to no longer include that players card in their hand.
* if a player tried to lay a phase, it will be checked to determine if it is congruent with the phase the player is trying to lay. If it is, the PhaseState object will be changed to reflect the player moving onto the next phase and the cards no longer in their hand.
* if a player tries to continue their or another players phase, the move will be checked to determine if the player has already laid a phase, is not skipped, and the card they are trying to lay actually continues the current phase being laid on.

The PhaseLocalGame constructor initializes its PhaseState object to be a brand new game. The canMove method returns true unless the game is over or we’re waiting exclusively on the other player. the checkIfGameOver method checks the PhaseState to see if the game has a player who has completed all 10 phases and is out of cards.

PhaseHumanPlayer

The purpose of the PhaseHumanPlayer is to allow a player to play a game via a GUI. Each PhaseHumanPlayer implements Animator and OnClickListener, and has a single AnimationSurfarface that it works with.

The PhaseHumanPlayer class defines the following instance variables:

* The state of the game, according to the last time the game was asked for it.
  + private PhaseState recentState;
* The activity that is responsible for the local game of the player.
  + private Activity myActivity;
* The surface on which the GUI for the player will be displayed.
  + private AnimationSurface surface;

The PhaseHumanPlayer is responsible for displaying the game for the human player in order for the player to see what is going on in the game. It also handles touch interactions.

PhaseHumanPlayer implements the following methods:

* **public void receiveInfo(GameInfo)** - sets the new state of the game, and allows for different actions to be done based on the information that was received
* **public Component setAsGiu()** - sets up the GUI layout and listeners
* **public void tick(Canvas canvas)**- performs animation
* **public void onTouch(MotionEvent me)** - handles touch events

(Other simple methods will be implemented as needed)

The tick method paints the Canvas to reflect the game’s state, and adjusts it by any non-committed moves the player may make, such as highlighting cards.

There will be several things that the user will be able to do that the Human player class will be responsible for that happen in the onTouch method.

* A user will be able to touch a card, and the card will become highlighted. The user may then choose what to do with the card by touching somewhere else on the screen, such as the discard pile or another players phase.
* A user will be able to touch the deck of cards to draw a card and this will cause a draw action to be sent to the game.
* A user will be able to rearrange their hand by long touching a card, and then touching another card in their hand which they wish to swap the cards place with.
* A user will be able to touch multiple cards, and all of them will become selected or highlighted and the user will then be able to touch on the main area of the screen (not in their hand) to lay a phase.

PhaseComputerPlayer and sublcasses

PhaseComputerPlayer is an abstract class and parent for each of the leaf AI classes. It is used to connect the AI to the the game framework.

PhaseComputerPlayer0 or the “Dumb Computer Player” is an AI for phase 10 that plays nearly as randomly as is possible within the rules of Phase 10.

* Start of turn / Draw
  + Randomly chooses between draw and discard pile if discard pile is a legal move otherwise just draws from draw pile
  + isDrawAction()
  + Creates and sends PhaseDrawAction to LocalGameState
  + Waits 2 seconds.
* Check for phase
  + If not phased yet and the phase exists in the hand then lay phase
  + Creates and sends PhaseLayPhaseAction if necessary to LocalGameState
  + Waits 2 seconds.
* Hit
  + If already phased choose a random card and a random pile and try to play that card on it
  + Creates and sends PhaseLayOnPhaseAction if necessary to LocalGameState
  + Waits 2 seconds.
* Discard
  + Discards a random card if it is a skip card randomly choose another player who is not currently skipped
  + Creates and sends PhaseDiscardAction or PhaseSkipAction to LocalGameState
  + Waits 2 seconds.

PhaseComputerPlayer1 or the “Smart Computer Player” is an AI for phase 10 that plays with the goal to win within the rules of Phase 10.

* Start of turn / Draw
  + If a set is needed and the number of sets need for phase has not been met and the card on top of the pile adds to it then draw that card
  + If a run is needed and is not currently met if the card on top of the discard pile adds to it draw it
  + If a number of the correct color is need and the card on top of the discard pile adds to it draw it
  + else draw from draw pile
  + Creates and sends PhaseDrawAction to LocalGameState
  + Waits 2 seconds.
* Check for phase
  + If not phased yet and the phase exists in the hand then lay phase
  + Creates and sends PhaseLayPhaseAction if necessary to LocalGameState
  + Waits 2 seconds.
* Hit
  + If already phased check all cards in hand with all possible hit locations and hit when possible until no more available or only one card in hand
  + Creates and sends PhaseLayOnStateAction if necessary to LocalGameState
  + Waits 2 seconds.
* Discard
  + If a skip card is in hand discard and skip another player who has skip currently phased with large bias on current phase favoring more and a small on their points favoring fewer.
  + If a card is not part of a phase or a part of a chance of a phase then discard it favoring 10-12 but otherwise choosing randomly
  + Else discard a card with the smallest part of a phase discard it favoring 10-12 but otherwise randomly
  + Creates and sends PhaseDiscardAction to LocalGameState
  + Waits 2 seconds.

**Walk-throughs of common operations**

There four different moves a player can make in the game:

* Draw a card
  + from discard pile
  + from draw pile
* Lay a phase
* Lay on a phase or hit
* Discard card
  + Discarding a skip card causes a new gui to popup to let the humanplayer choose which player is skipped

These actions can unfold in several different ways for the human player. The human player is assumed to be taking the turn unless otherwise noted.

* Take a turn make phase action, and discard a skip card but no hit action
* Take a turn, hit and then go out
* Other player has skipped the human player
* Other player goes out

Scenario 1: Take a turn and make no hit or phase action

* The turn starts and a new state is sent to all players.
  + The UI is refreshed to display information about the new state
* The human player then draws
  + The User chooses between drawing from the draw pile and the discard pile by touching one of them and the PhaseDrawAction() is sent to the game
    - If the discard pile was drawn from the UI updates to show the card below the top card on the discard pile
    - If the draw pile was drawn from and there are no more cards in the draw pile the UI updates to reflect this
  + The drawn card is added to the hand and the UI is updated to reflect this
  + The player is then allowed to lay phase if they have not phased, hit if the have phased (but this is not done in this scenario), or discard a card
* The new state is sent to all players
* The human player then discards a skip card
  + The User touches a card from his or her hand, which will become highlighted, and then touches the discard pile.
    - A GUI pops up and asks the player to choose a player to skip
  + A PhaseSkipAction is then sent to the game
* The new state is sent to all players
* The player’s move is over and the game allows the next person to make a move

Scenario 2: Player Phases

* The turn starts and a new state is sent to all players.
  + The UI is refreshed to display information about the new state.
* The human player then draws.
  + The User chooses between drawing from the draw pile and the discard pile by touching one of them and the PhaseDrawAction() is sent to the game
    - If the discard pile was drawn from the UI updates to show the card below the top card on the discard pile
    - If the draw pile was drawn from and there are no more cards in the draw pile the UI updates to reflect this
  + The drawn card is added to the hand and the UI is updated to reflect this
  + The player is then allowed to lay phase if they have not phased, hit if the have phased (but this is not done in this scenario), or discard a card
* The new state is sent to all players
* The player then selects cards to lay down as a phase.
  + The user touches a card in her hand and it will be come highlighted. The user can then touch multiple cards to highlight all of the cards he or she wishes to lay down in their phase.
  + The user then touches on the main board area where their phase will be displayed in order to lay the phase.
  + A PhaseLayPhaseAction will then be sent to the game.
* The new state is then sent to all players.
* The Human player then discards a card.
  + The User touches a card from his or her hand, which will become highlighted, and then touches the discard pile.
  + A PhaseDiscardAction is then sent to the game.
* The new state is sent to all players
* The player’s move is over and the game allows the next person to make a move

Scenario 3: Player hits and goes out

* The turn starts and a new state is sent to all players.
  + The UI is refreshed to display information about the new state.
* The human player then draws. (As described in Scenario 1 & 2)
* The new state is sent to all players
* The user selects a card to lay on a phase and plays it
  + The user touches the card that he or she wishes to lay on a phase.
  + The user then touches the phase they wish to lay the card on.
  + The user will be asked in a message which subPhase the user wishes to play on, and the user will pick which subPhase.
  + If the phase chosen is a run, the user will then be asked if they wish to play on the top or the bottom of the phase.
  + Then a PhaseLayOnPhaseAction will be sent to the game.
* The new state is then sent to all players.
* The user then selects another card (their last card) to lay on a phase and plays the card ( the same actions as before occur)
* A message then appears to the user saying that the round is complete and displays the information of that round, including the winner of the round, which is the current user, the phase other users are on, and the number of points that each user has.
* The user then exits this screen by touching outside of the message box and the next round of the game will start.

Scenario 4: Computer player skips a Human player

This scenario starts at the beginning of a computer players turn.

* The computer will draw a card.
  + Depending on what the computer decides, a PhaseDrawAction will be sent to the game either with, isDeck as true or false. For this scenario that value will be true.
* The new state is then sent to all players.
* The computer player then chooses to discard a skip card.
  + The computer player will send a PhaseSkipAction to the game, in this scenario, having choosen the Human player to skip.
* The new state is then sent to all players.
* The human player who was skipped receives a message on his or her screen informing the user that a skip card has been played on them and who played it on them.
* Other users take their turns until it reaches the skipped human player
  + The human player who was skipped will receive a message when it is their turn and they are being skipped. The message will say something along the lines of “Your turn has just been skipped.”
* The user then will wait for all other players to take their turns before he or she may play.

**Test Plan**

The team will use test driven development by using the JUnit testing framework to ensure the correctness of each class. The entire program will be debugged through user testing which includes playing the game normally as well as trying unexpected interactions.

JUnit test work through a series up setup steps to design each unit test.

* Use constructor that takes in an encoded string to set up states
* Use method to export states to strings to capture current state
* Use JUnit tests to test classes individually without reliance on uncompleted classes

Testing timeline

* Start testing with PhaseState object
* Test PhaseMoveAction class and its descendents
* Test PhaseLocalGame
* Test PhaseHumanPlayer
* Test PhaseComputerPlayer

**Plan for Implementing the Game / Division of Labor**

* Complete Design Document - Team (October 31st)
* Human Player and Layouts - Justice and Hunter (November 7th)
* Local Game and State - Danny and Max (November 14th)
* Move Actions - Danny and Max (November 17th)
* Computer Player - Justice (November 19th)
* Main Activity - Hunter (November 19th)
* Alpha Release (Test-to-Succeed) - Team (November 19th)
* Beta Release (Test-to-Fail) - Team (December 1st)
* Release Candidate - Team (December 5th)
* Final Bug Validation - Team (December 11th)