Phase 10 Design

Danny Nelson, Max Robinson, Justice Nichols, and Hunter Torset

Phase 10 Overview

- 2 to 6 players
- Uses 2 standard card decks or Phase 10 deck
- Game contains 10 phases players must complete
- Objective of game is to be the first player to complete all 10 phases with the least points

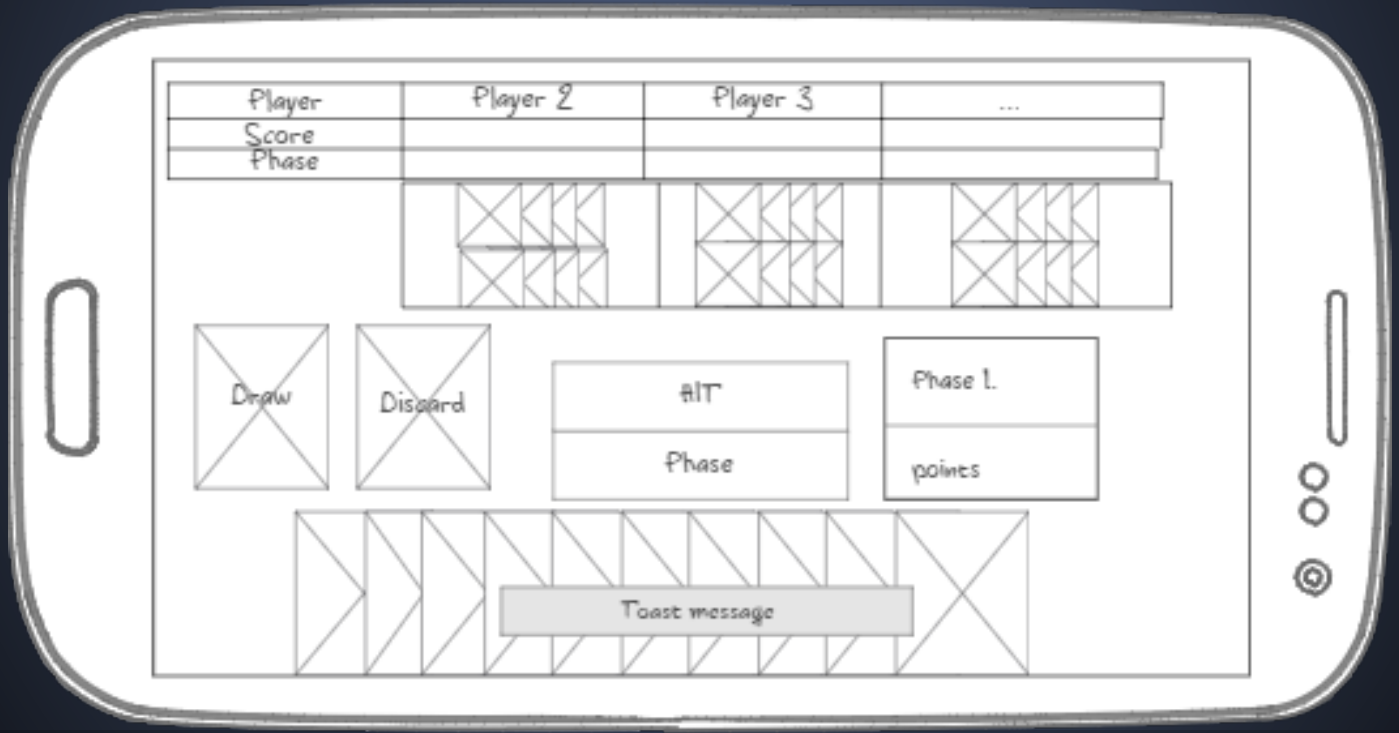
Phase 10 Overview Cont.

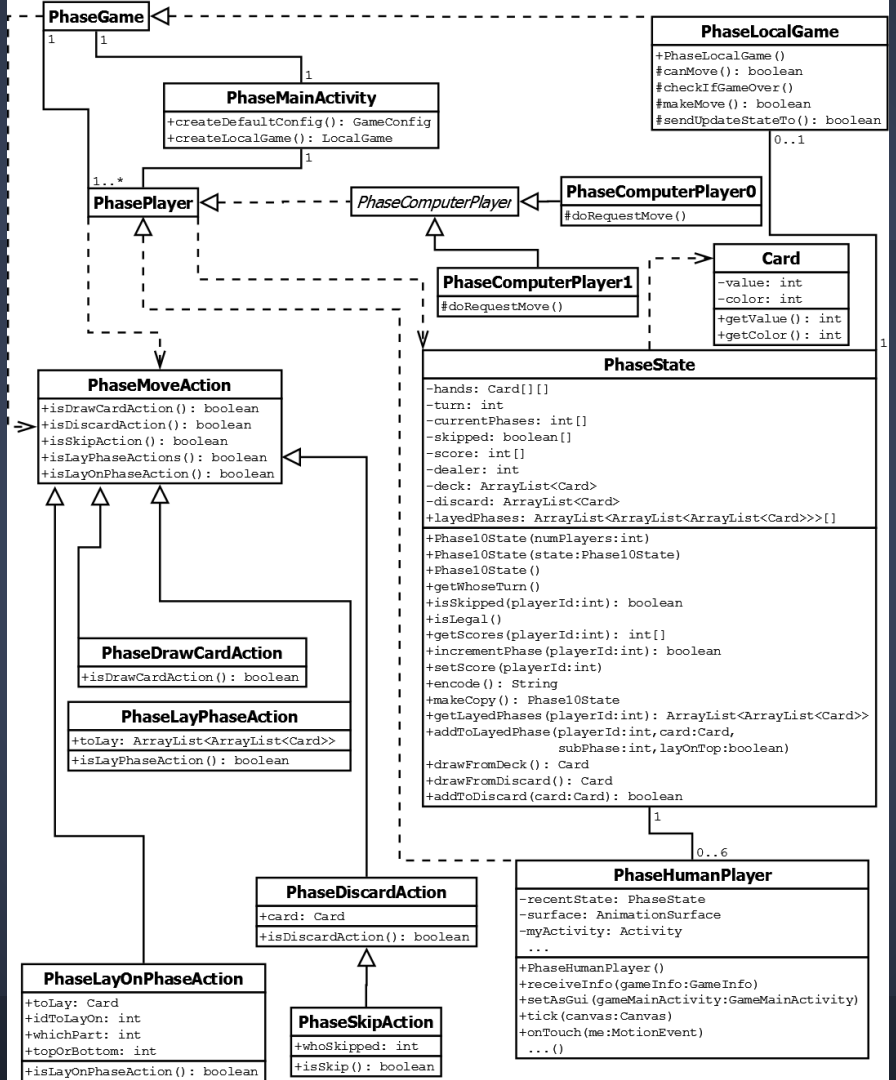
- Players are dealt 10 cards and begin on phase 1
- Players lay down cards when they have completed current phase
- Once phase has been laid, try to continue own or another players phase

Phase 10 Overview Cont.

- Players go out by laying all cards down
- At the end of round, card values remaining in other players hands are tallied and added to score
- In the event of a tie, player with the lowest score wins

Phase 10 GUI


Player
Player 2
Player 3
...
Score
Phase
Draw
Discard
AIT
Phase
Phase 1.
points
Toast message


PhaseGame
PhaseLocalGame
+PhaseLocalGame()
#canMove(): boolean
#checkIfGameOver()
#makeMove(): boolean
#sendUpdateStateTo(): boolean
PhaseMainActivity
+createDefaultConfig(): GameConfig
+createLocalGame(): LocalGame
PhasePlayer
PhaseComputerPlayer
PhaseComputerPlayer0
#doRequestMove()
PhaseComputerPlayer1
#doRequestMove()
Card
-value: int
-color: int
+getValue(): int
+getColor(): int
PhaseMoveAction
+isDrawCardAction(): boolean
+isDiscardAction(): boolean
+isSkipAction(): boolean
+isLayPhaseActions(): boolean
+isLayOnPhaseAction(): boolean
PhaseState
-hands: Card[][]
-turn: int
-currentPhases: int[]
-skipped: boolean[]
-score: int[]
-dealer: int
-deck: ArrayList<Card>
-discard: ArrayList<Card>
+layedPhases: ArrayList<ArrayList<ArrayList<Card>>>[]
+Phase10State(numPlayers:int)
+Phase10State(state:Phase10State)
+Phase10State()
+getWhoseTurn()
+isSkipped(playerId:int): boolean
+isLegal()
+getScores(playerId:int): int[]
+incrementPhase(playerId:int): boolean
+setScore(playerId:int)
+encode(): String
+makeCopy(): Phase10State
+getLayedPhases(playerId:int): ArrayList<ArrayList<Card>>
+addToLayedPhase(playerId:int,card:Card,
subPhase:int,layOnTop:boolean)
+drawFromDeck(): Card
+drawFromDiscard(): Card
+addToDiscard(card:Card): boolean
PhaseDrawCardAction
+isDrawCardAction(): boolean
PhaseLayPhaseAction
+toLay: ArrayList<ArrayList<Card>>
+isLayPhaseAction(): boolean
PhaseDiscardAction
+card: Card
+isDiscardAction(): boolean
PhaseHumanPlayer
-recentState: PhaseState
-surface: AnimationSurface
-myActivity: Activity
...
+PhaseHumanPlayer()
+receiveInfo(gameInfo:GameInfo)
+setAsGui(gameMainActivity:GameMainActivity)
+tick(canvas:Canvas)
+onTouch(me:MotionEvent)
...()
PhaseLayOnPhaseAction
+toLay: Card
+idToLayOn: int
+whichPart: int
+topOrBottom: int
+isLayOnPhaseAction(): boolean
PhaseSkipAction
+whoSkipped: int
+isSkip(): boolean
1
1
1
1..*
0..1
1
1
0..6

PhaseMoveAction

- Abstract class that all other move action classes will subclass
- Contains methods that will be overridden by subclasses based on which move action they are.

PhaseDrawCardAction

- This action is triggered when a user draws a card from the deck.
- Has no instance variables, one override method.
- isDrawCardAction() is overridden to return true.

PhaseDiscardAction

- This action is triggered when a user discards a card from their hand.
- Has one instance variable and one override method.
- private Card card; The card the user wishes to discard.
- isDiscardAction(); return true.

PhaseSkipAction

- This action is triggered when the user discards/plays a skip card.
- Subclasses PhaseDiscardAction.
- Has one additional instance variable and one override method.
  - int whoSkipped ; The ID of the player being skipped
- isSkipAction(); return true;

PhaseLayPhaseAction

- This action is triggered when a user wants to lay a phase for the round.
- Has one instance variable and one override method.
- ArrayList<ArrayList<Card>>; Contains that cards that are to be laid as the phase.
- isLayPhaseAction(); return true;

PhaseLayOnPhaseAction

- This action is triggered when a user plays a card on a phase that has already been laid down by a player.
- Has four instance variables and one override method
  - Card toLay; int idToLayOn; int whichPart; int topOrBottom;
- isLayOnPhaseAction(); return true;

PhaseState Data

- 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;`

PhaseState Data cont.

- 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>>>[]`

PhaseState Non-Modification Methods

- **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(int playerID)** - the current score the the player whose ID is specified
- **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 separated by commas, denoting the value of variables in the order listed above.
- **ArrayList<ArrayList<Card>> getLayedPhase(int playerID)** - the current cards in the player's hand whose ID is specified.

PhaseState Modification Methods

- **void 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 playerID, Card card, int subPhase, boolean layOnTop) -** allows players to hit on another player's layed phases. The playerID will play the specified card on.
- **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.

PhaseLocalGame

- This class is the version of the game that all players will have an instance of that contains information for that player specifically.
- Has no instance variables and four protected methods.

PhaseLocalGame (cont.)

- protected boolean canMove();
  - Tells whether that player can move right now.
- protected checkIfGameOver();
- protected makeMove();
  - This is where the local game will do most of the work of letting a player make a move in the game.
- protected boolean sendUpdateStateTo();
  - This will send the updates to the game for changes to the game state to be made.

Card

- This Class holds the information that would usually be on a physical card.
- Has two private instance variables.
  - int value; int color;
- Has two public methods.
  - int getValue(); int getColor();

Sequence Diagram: Human Player Makes a Move

- Right before turn begins: Game sends the new state to all players.
  - Human Player’s GUI updated
- Human touches the deck to draw a card
- Creates a drawAction which is check for validity by LocalGame. DrawAction is then sent to the game.
- The game updates the state of the game and then sends the updated state to all players.
  - Human Player’s GUI updated
- Human player has the option to touch and drag cards in their hand to rearrange the cards in their hand.
  - This is all handled by the Human Player class and GUI changes.
  - This can be done as many times the user wishes.

Sequence Diagram: (cont.)

- The Human Player selects a card to discard by dragging a card from their hand to the discard pile.
- The Local Game checks what card is being discarded. Assume it is not a skip card. A discardAction is then made and checked if it is a valid Action and is then sent to the game
- Game processes the discardAction, and updates the state.
- Game sends the updated state to all players.
    - Human GUI is updated to reflect the changes in state.
- The human players turn is now over and they will not be allowed to make any further actions until their next turn.

“Dumb” Computer Player

- Inherits from computer player
  - Computer player has no GUI functionality and has a simple algorithmic response to the current situation of the game.
  - Has one method DoRequestMove( )
  - Follows the following rules

“Dumb” Computer Player (cont.)

- Start of turn / Draw
  - If the top of the discard is not a skip the computer randomly chooses between draw and discard pile
  - Else draws from draw pile
- Check for phase
  - If not phased yet and the phase exists in the hand then lay phase
- Try to Hit
  - If already phased choose a random card and a random pile and try to play that card on it
- Discard a card
  - Discards a random card if it is a skip card randomly choose another player who is not currently skipped (i.e. not skipped in same round)

“Smart” Computer Player

- Inherits from computer player
  - Computer player has no GUI functionality and has a complex algorithmic response to the current situation of the game.
  - Has one method DoRequestMove( )
  - Follows the following rules

“Smart” Computer Player

- 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
- Check for phase
  - If not phased yet and the phase exists in the hand then lay phase
- Try to Hit
  - If already phased check all cards in hand with all possible hit locations and hits when possible until no more available or only 1 card in hand
- Discard a card
  - 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.
  - Else 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 if all cards are part of a possible phase then remove a card with the smallest part of a phase discard it favoring 10-12 but otherwise randomly

Project Plan - Timeline and Assignments - Team

- Oct. 27th, Design Review.
- Oct. 31st, Complete Design Due.
- Nov. 7th, Dummy UI Complete (layouts done).
- Nov. 14th, Game Logic Complete.
- Nov. 18th, UI Connected to Game Logic.
- Nov. 19th, Test-to-Succeed (Alpha Version).
- Dec. 1st, Test-to-Fail (Beta Version).
- Dec. 5th, Release Version.
- Dec. 11th, Bug Validation.

Project Plan - Timeline and Assignments - Specific

- Danny and Max - Local Game and State (November 14th)
- Justice - Computer Player (November 19th)
- Move Actions - Danny and Max (November 17th)
- Human Player and Layouts - Justice and Hunter (November 7th)
- Main Activity - Hunter (November 19th)

Project Plan - Implementation

- 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

Project Plan - Testing

- Start testing with PhaseState object
- Test PhaseMoveAction class and its descendents
- Test PhaseLocalGame
- Test PhaseHumanPlayer
- Finally, test PhaseComputerPlayer