23

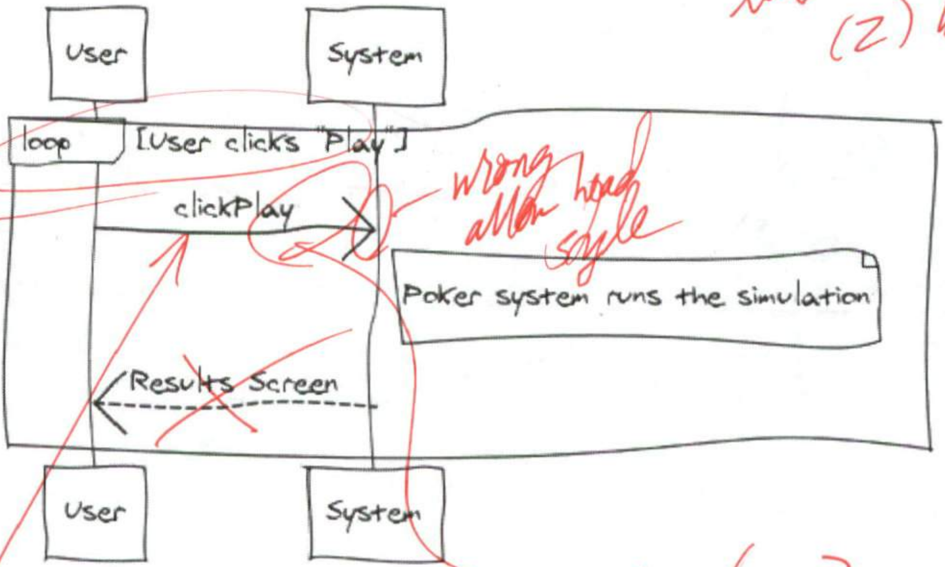
CPP Please add
- cover page
- short textual description of each figure
- domain model

SSD for use case 1

use case title

```
loop User clicks "Play"
   User->System: clickPlay
   note right of System: Poker system runs the simulation
   System-->User: Results Screen
End
```

If you must include this in the document, at least (1) label it, (2) use a smaller font & (3) put it after the drawing, if possible.



wrong arrow head style

loop its conditions

any parameters? e.g., number of players parallel vs. sequential

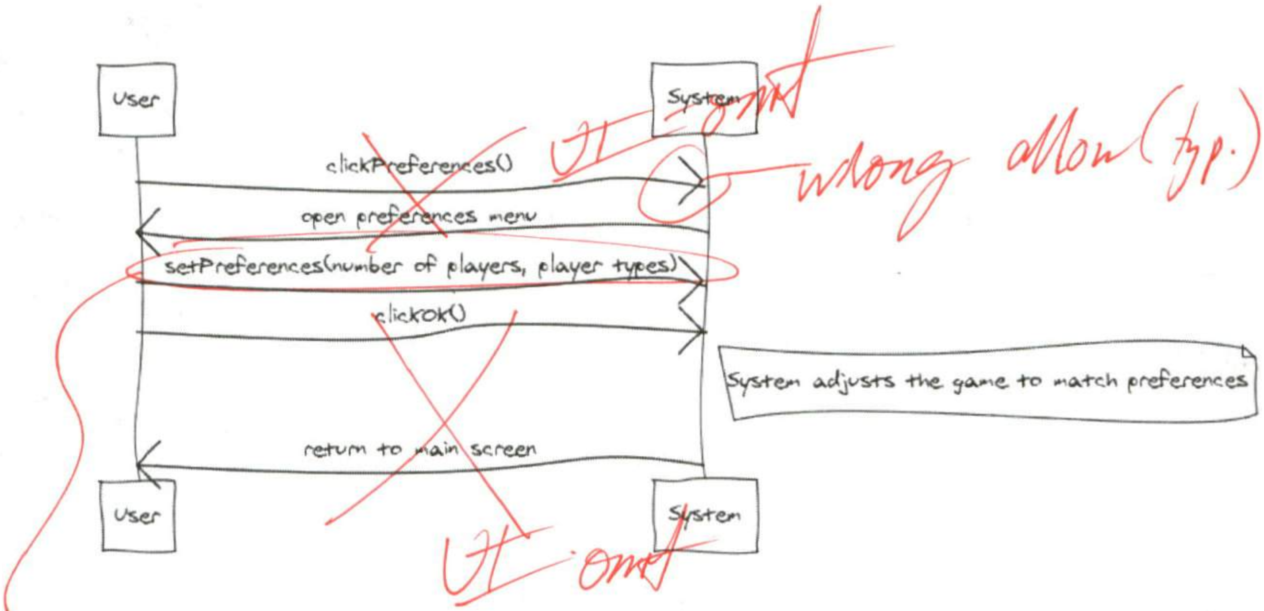
Or perhaps expand the note to say that the simulation runs using the current preferences.

SSDs should not be in terms of the UI.

You're trying to identify the system operations coming into the domain layer, not to draw UI-focused use cases in a new format

SSD for use case 2

User->System: clickPreferences()
System->User: open preferences menu
User->System: setPreferences(number_of_players, player_types)
note right of System: System adjusts the game to match preferences
System->User: return to main screen


User
System
UI - omit
wrong arrow (typ.)
~~clickPreferences()~~
~~open preferences menu~~
setPreferences(number of players, player types)
~~clickOK()~~
System adjusts the game to match preferences
~~return to main screen~~
User
System
UI - omit

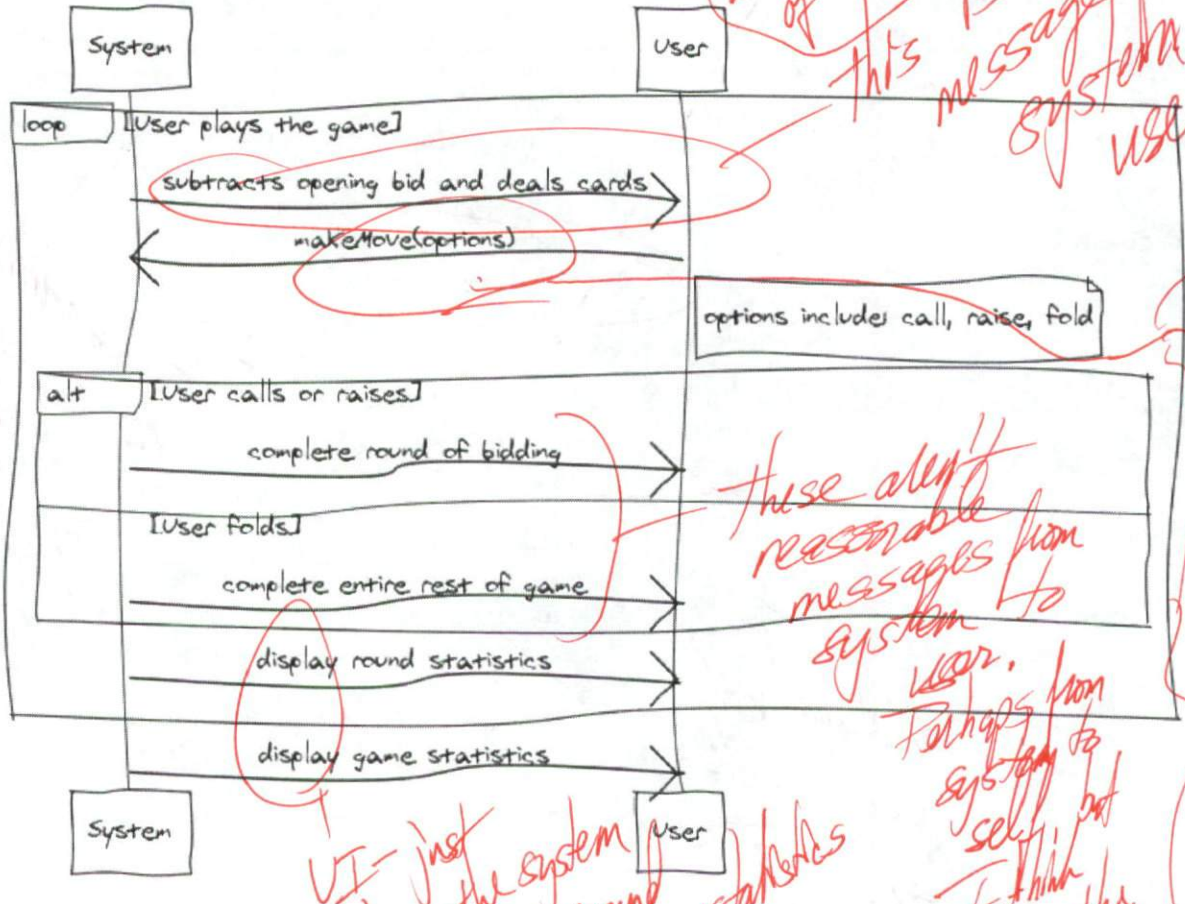
this is a decent system operation. It would be better if there was slightly more detail on what "playertypes" means.

SSD for use case 3

```
loop User plays the game
System->User: subtracts opening bid and deals cards
User->System: makeMove(options)
note right of User: options include: call, raise, fold
alt User calls or raises
   System->User: complete round of bidding
else User folds
   System->User: complete entire rest of game
end
System->User: display round statistics
end
System->User: display game statistics
```

I'm not wild about having the System in the active roll here. I think you want the user in the active roll w/ assumptions about the pre-state of the system

this is a message from system to user?



these aren't reasonable messages from system to user. Perhaps from system to self but I think inverting the control will help decide.

I think you want three alternatives here w/ three different system operations ~~message~~

UI- just show the system returning round & game statistics see notes on problems w/ SSD

Contract CO1: clickPlay

Operation:	clickPlay()
Cross References:	Use Cases: Starting the Game
Preconditions:	User has set player options
Postconditions:	The user watches the simulation take place

Contract CO2: setPreferences

( ) parameters

Operation:	setPreferences()
Cross References:	Use Cases: Setting Player Options
Preconditions:	The application has initialized properly Parameter menus are functioning properly
Postconditions:	The types of players have changed based on what the user has selected.

Contract CO3: playGame

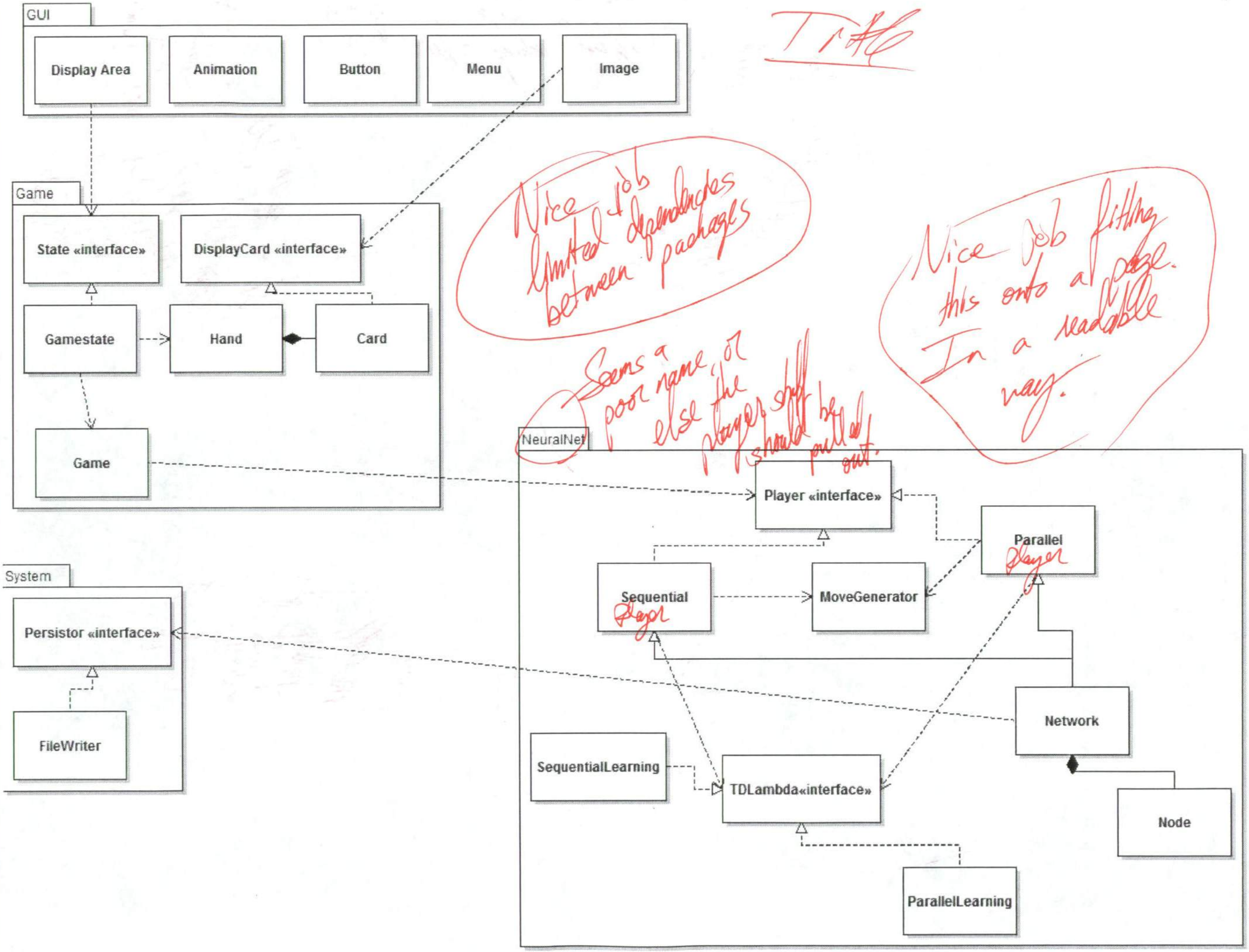
Operation:	playGame()
Cross References:	Use Cases: N/A
Preconditions:	User has set player options to include human player User has clicked "Start" button
Postconditions:	User views game statistics and/or plays another game

Post conditions must be (1) Past tense (2) In terms of changes to the domain model. (typ.

Contract CO4: makeMove

Operation:	makeMove()
Cross References:	Use Cases: N/A
Preconditions:	The game is in play It is the user's turn in the game
Postconditions:	The appropriate amount of chips move to the pot The user's stack is deducted the appropriate amount The game continues on to the next turn

parameters? see notes on SSDs



Nice job limited dependencies between packages

Nice job fitting this onto a page. In a readable way.

Seems a poor name, or else the player stuff should be pulled out.

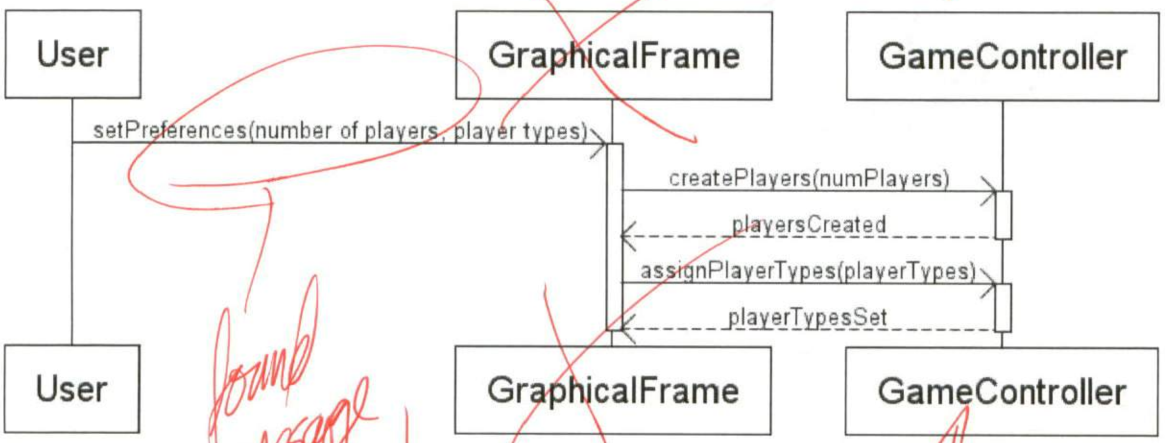
Set Preferences Operation

see page 1

participant User
participant GraphicalFrame
participant GameController
User -> GraphicalFrame: setPreferences(number of players, player types)
activate GraphicalFrame
GraphicalFrame -> GameController: createPlayers(numPlayers)
activate GameController
GameController --> GraphicalFrame: playersCreated
deactivate GameController
GraphicalFrame -> GameController: assignPlayerTypes(playerTypes)
activate GameController
GameController --> GraphicalFrame: playerTypesSet
deactivate GameController
deactivate GraphicalFrame

No UI on interaction diagrams; we're just modeling the domain.

Perhaps ~~the~~ the domain layer doesn't store objects at all; but gets them in the playGame sys. op.


User
GraphicalFrame
GameController
setPreferences(number of players, player types)
createPlayers(numPlayers)
playersCreated
assignPlayerTypes(playerTypes)
playerTypesSet
User
GraphicalFrame
GameController

found message should target the controller object

does game controller store the preferences itself? Should it create/~~mutate~~ objects to store them?

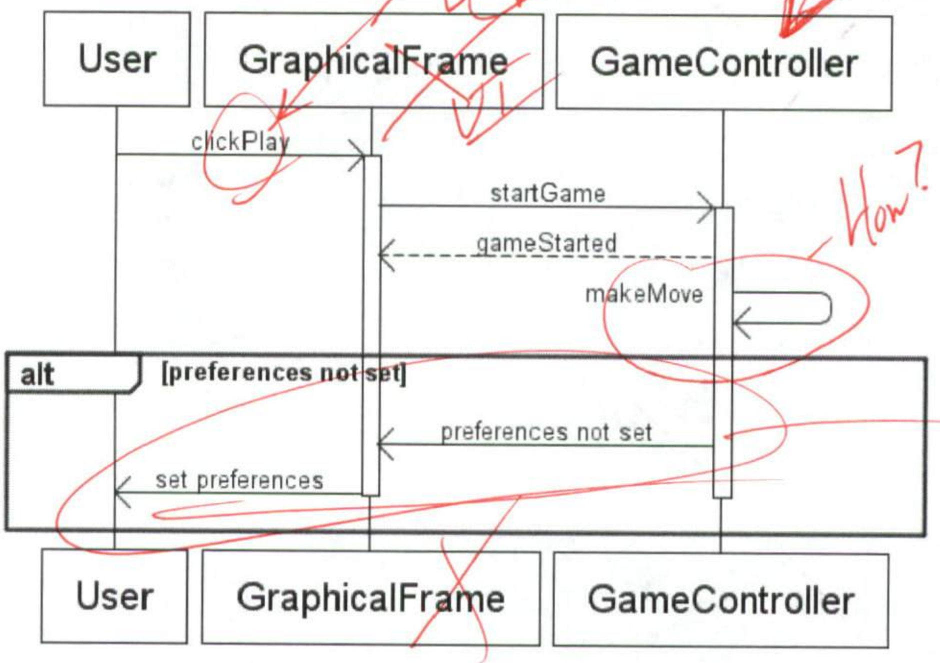
Begin Game Operation

```
participant User
participant GraphicalFrame
participant GameController
User -> GraphicalFrame: clickPlay
activate GraphicalFrame
GraphicalFrame -> GameController: startGame
activate GameController
GameController --> GraphicalFrame: gameStarted
GameController -> GameController: makeMove
alt preferences not set
GameController -> GraphicalFrame: preferences not set
GraphicalFrame -> User: set preferences
deactivate GameController
deactivate GraphicalFrame
```

playGame is much better. Do you see why?

Bad name

What objects does the GameController have to interact w/? that's the whole point of interaction diagrams.



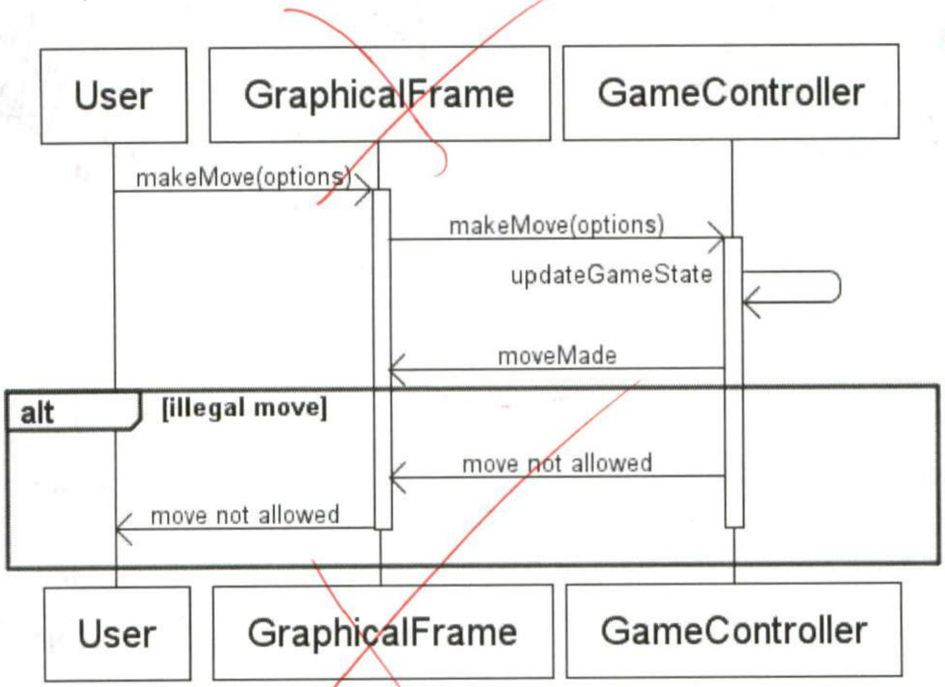
How?

This is a strong argument for having the user interface store the preferences & to pass them in with the playGame sys. op.

Make Move Operation

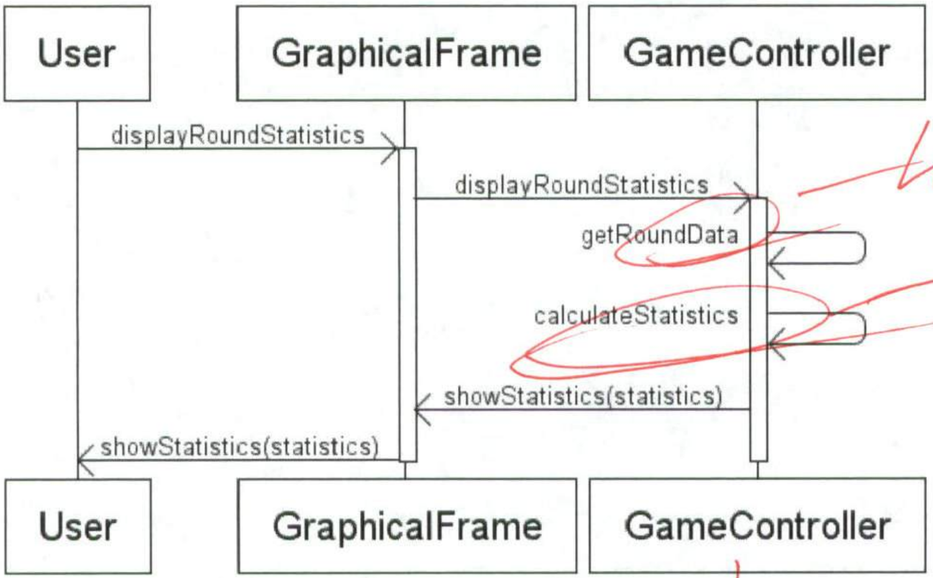
```
participant User
participant GraphicalFrame
participant GameController
User -> GraphicalFrame: makeMove(options)
activate GraphicalFrame
GraphicalFrame -> GameController: makeMove(options)
activate GameController
GameController -> GameController: updateGameState
GameController -> GraphicalFrame: moveMade
alt illegal move
GameController -> GraphicalFrame: move not allowed
GraphicalFrame -> User: move not allowed
deactivate GameController
deactivate GraphicalFrame
```

see notes on SSD, OC, & on previous page



Display Round Statistics Operation

participant User
participant GraphicalFrame
participant GameController
User -> GraphicalFrame: displayRoundStatistics
activate GraphicalFrame
GraphicalFrame -> GameController: displayRoundStatistics
activate GameController
GameController -> GameController: getRoundData
GameController -> GameController: calculateStatistics
GameController -> GraphicalFrame: showStatistics(statistics)
GraphicalFrame -> User: showStatistics(statistics)
deactivate GameController
deactivate GraphicalFrame



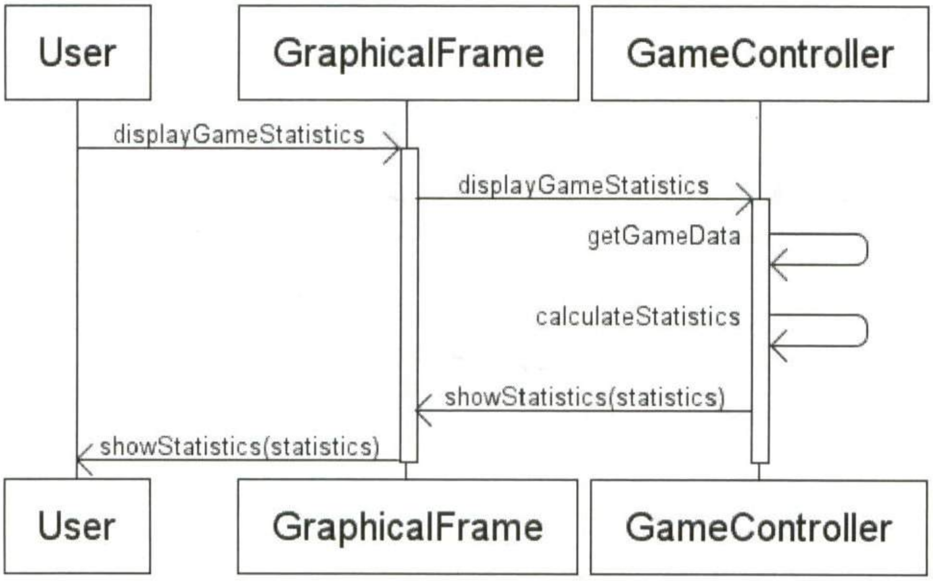
How?

How?

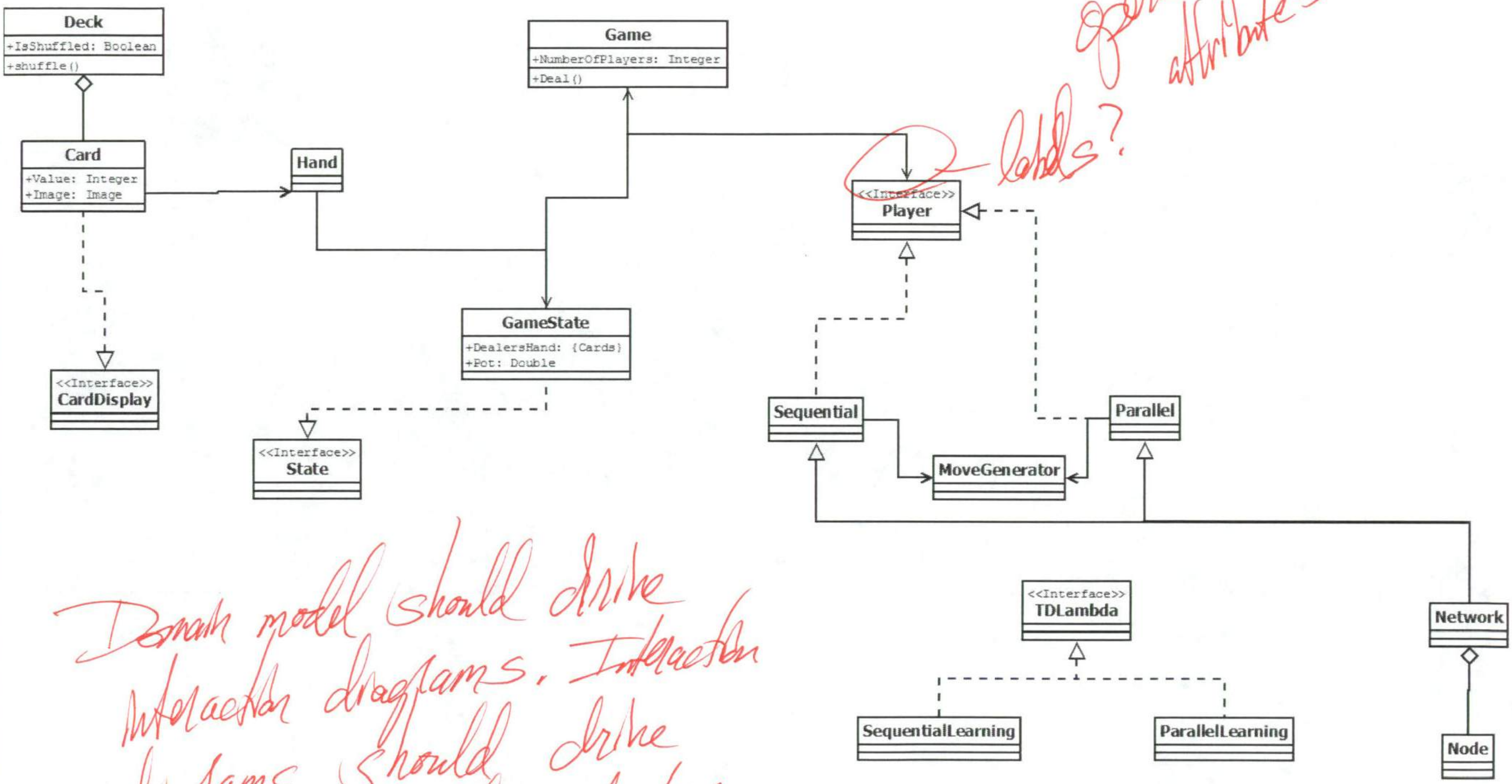
Your game controller is just one giant uber-class according to these drawings.
⇒ terrible cohesion.

Display Game Statistics Operation

```
participant User
participant GraphicalFrame
participant GameController
User -> GraphicalFrame: displayGameStatistics
activate GraphicalFrame
GraphicalFrame -> GameController: displayGameStatistics
activate GameController
GameController -> GameController: getGameData
GameController -> GameController: calculateStatistics
GameController -> GraphicalFrame: showStatistics(statistics)
GraphicalFrame -> User: showStatistics(statistics)
deactivate GameController
deactivate GraphicalFrame
```



See previous page



operations?
attributes?
labels?

Domain model should drive interaction diagrams. Interaction diagrams should drive design class diagram.