Assignment 4

Alex Geenen, Bart de Jonge, Mark van de Ruit, Menno Oudshoorn, Stefan de Vringer

Software Engineering Methods, Group 1

Exercise 1: Your wish is my command, reloaded

For this exercise, we first show the new requirements for the new features we wished to develop. Due to the nature of this exercise, we have no actual functional requirements a user could express. We spent the entire exercise refactoring and improving existing code, and fixing concurrency issues.

Functional Requirements

Must haves: none. Should haves: none. Could haves: none. Won't haves: none.

Non-functional Requirements

- The game will not suffer from crashes due to concurrency issues anymore, as these make the multiplayer almost (or completely) unplayable.
- The game will use a queue, or similar, when processing host/client influences on a game, so as to prevent synchronisation issues.
- The game's code must be more logical in order of organisation, layout and naming. Reexamination of existing code is advised.

Now that we have discussed the requirements for exercise 1, we show how we have continued using Responsibility Driven Design to further develop these features. We do this by creating CRC cards for new and updated classes. We have also kept our (rather humongous) UML class diagram up to date. **Due to the size of this diagram, however, it should be opened separately from this document. See A4-UML.png.**

On the subject of solving concurrency issues we have a few remarks. The problems were caused by the host and client adding and removing elements from lists that other parts of the game where iterating over. To prevent this we implemented a combination of the command and singleton pattern. When the host or client wants to add or delete an element from a list, we schedule this using a command. We add this command to a concurrency safe queue which is emptied at the very end of the gameState.update() method. This ensures that no other part of the program is busy with the lists at that moment and will thus solve our concurrency problems. The queue is embedded in a singleton class called CommandQueue. Because this class is a singleton every class of the game will add his commands to the same queue. A UML-diagram and sequence diagram are included to further visualize this solution.

Furthermore, we have split the reexamination (and possible refactoring) into a separate spreadsheet-document, where we assign tasks to each of us, make notes on possible refactors, and confirm what kind of refactor has occurred. **Due to the size of this spreadsheet, it should be opened separately from this document. See A4-Refactor.pdf.** Please note that some team members, Menno and Stefan, have performed most if not all of the refactoring tasks, as all other team members were occupied battling god-classes.

It should be noted that we have created CRC cards for all new/updated classes in Exercise 2 and that these are visible here as well.

To clarify: CRC cards contain the class name, superclass, and subclasses. Then the bottom-left part of a crc card describes responsibilities, while the bottom-right part describes collaborations.

Name: Player	
Superclass: none	
Subclasses: none	
Represent and manage all logic and rendering for a Player in the game	All of its attached Player***Helper classes.

Name: PlayerMovementHelper	
Superclass: none	
Subclasses: none	
Manage all logic and drawing for Player movement	Player and all its helper classes, Gate, Image.

Name: PlayerPowerupHelper	
Superclass: none	
Subclasses: none	
Manage all logic and drawing for Player powerups	Player and all its helper classes, Powerup, Image

Name: PlayerWeaponHelper
Superclass: none

Subclasses: none	
Manage all logic and drawing for Player weapons	Player and all its helper classes, Powerup, Weapon, Image

Name: PlayerGateHelper	
Superclass: none	
Subclasses: none	
Manage all logic and drawing for Player interaction with gates	Player and all its helper classes, Gate.

Name: PlayerLogicHelper	
Superclass: none	
Subclasses: none	
Manage all miscellaneous logic and drawing for Player.	Player and all its helper classes, Image.

Name: GameState	
Superclass: BasicGameState	
Subclasses: none	
Manage the main gameplay loop and rendering.	All GameStateHelper-extended classes, Player, Host, Client, RND.
Manage the pause menu.	GameStatePauseHelper, RND, Button.

Name: GameStateHelper	
Superclass: none	
Subclasses: GameStateCirclesHelper, -GateHelper, -InterfaceHelper, -ItemsHelper, -LevelsHelper, -LogicHelper, -PauseHelper, -PlayerHelper	
Manage offsprings of the main gameplay loop and rendering.	All GameStateHelper-extended classes.

Name: GameStateCirclesHelper	
Superclass: GameStateHelper	
Subclasses: none	
Manage GameState logic and drawing concerning circles	GameState, other helper classes, BouncingCircle, CircleList, RND.

Name: GameStateGateHelper	
Superclass: GameStateHelper	
Subclasses: none	
Manage GameState logic and drawing concerning gates	GameState, other helper classes, Gate, RND.

Name: GameStateInterfaceHelper	
Superclass: GameStateHelper	
Subclasses: none	
Manage GameState logic and drawing	GameState, other helper classes, Image,

concerning the UI.	RND.

Name: GameStateItemsHelper
Superclass: GameStateHelper
Subclasses: none

Manage GameState logic and drawing concerning powerups and coins	GameState, other helper classes, Powerup, Coin, Image, RND.

Name: GameStateLevelsHelper
Superclass: GameStateHelper
Subclasses: none

Manage GameState logic concerning levels	GameState, other helper classes, MyRectangle, Level.

Name: GameStateLogicHelper
Superclass: GameStateHelper
Subclasses: none

Manage GameState logic concerning miscellaneous events such as time.	GameState, other helper classes.

Name: GameStatePauseHelper

Superclass: GameStateHelper	
Subclasses: none	
Manage GameState logic and drawing concerning the pause menu	GameState, other helper classes, Button, RND.

Name: GameStatePlayerHelper	
Superclass: GameStateHelper	
Subclasses: none	
Manage GameState logic and drawing concerning the player	GameState, other helper classes, WeaponList, Weapon, Player, RND.

Name: RenderObject	
Superclass: none	
Subclasses: none	
Manage flexible data storage so RND doesn't get cluttered when drawing objects.	Graphics, RND, Image

Name: Client
Superclass: Connector

Subclasses: none	
Manage communication with another game during multiplayer, client-sided.	GameState, all Client helper classes, ClientMessageReader, Logger.

Name: ClientMessageReader	
Superclass: none	
Subclasses: none	
Process incoming messages received by a client.	GameState, Client, all Client helper classes, Logger.

Name: ClientPowerupsHelper	
Superclass: none	
Subclasses: none	
Process incoming messages received by a client, aimed at powerups.	GameState, Client, ClientMessageReader, Logger.

Name: ClientCoinsHelper	
Superclass: none	
Subclasses: none	
Process incoming messages received by a client, aimed at coins.	GameState, Client, ClientMessageReader, Logger.

Name: Command	
Superclass: none	
Subclasses: AddDroppedCoinCommand, - PowerupCommand, -FloatingScoreCommand, -PowerupToPlayerCommand, RemoveCircleCommand, -DroppedCoinCommand, DroppedPowerupCommand, -FloatingScoreCommand, -SpeedPoweruCommand, SetCircleListCommand	
Form abstract basis for all Commands. Commands are small classes holding a small item of code, which will influence the game in one way or another.	GameState, Player, Client, Host, Connector, CommandQueue

Please note that, due to the sheer ridiculous number of Command subclasses (most of which are no more than five lines of code), these will not get separate CRC cards. The above card should give enough information as to their purpose.

Name: CommandQueue	
Superclass: none	
Subclasses: none	
Queue and process incoming commands so as to prevent synchronisation issues.	Command.

Exercise 2: Software Metrics

For this assignment, we had to pick the first free Incode-detected design flaws in order of severity. After running incode, we conclude these three are the most severe flaws:

1. God classes
2. Schizofrenic classes
3. Internal duplications

We will be addressing these flaws in the next parts of this assignment.

2.1:

The analysis file of our project (before any modifications were made) is bundled with this document (in the same folder) and is named **Bubble Trouble_1444840554097.result**. Please refer to this file for this part of the assignment.

2.2, God Classes:

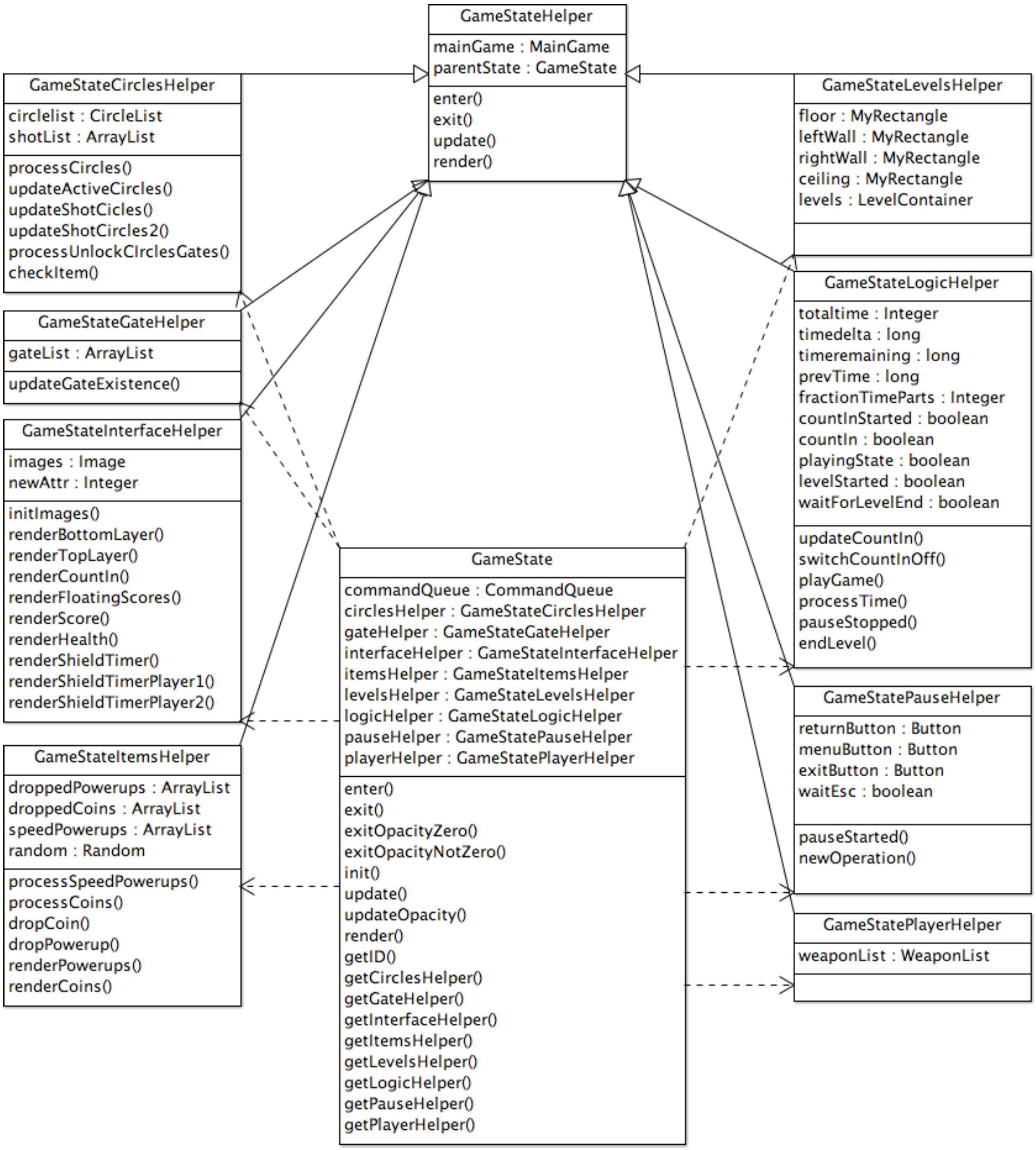
Due to the lacking/incomplete functionality in the early stages of the game, this functionality was placed in a few classes. The error in design that we made afterwards, was putting all added functionality in these same classes, which caused these classes to swell, with many complex methods and extensive invocation of methods from other classes. This functionality should have been split into multiple classes or redistributed between existing classes.

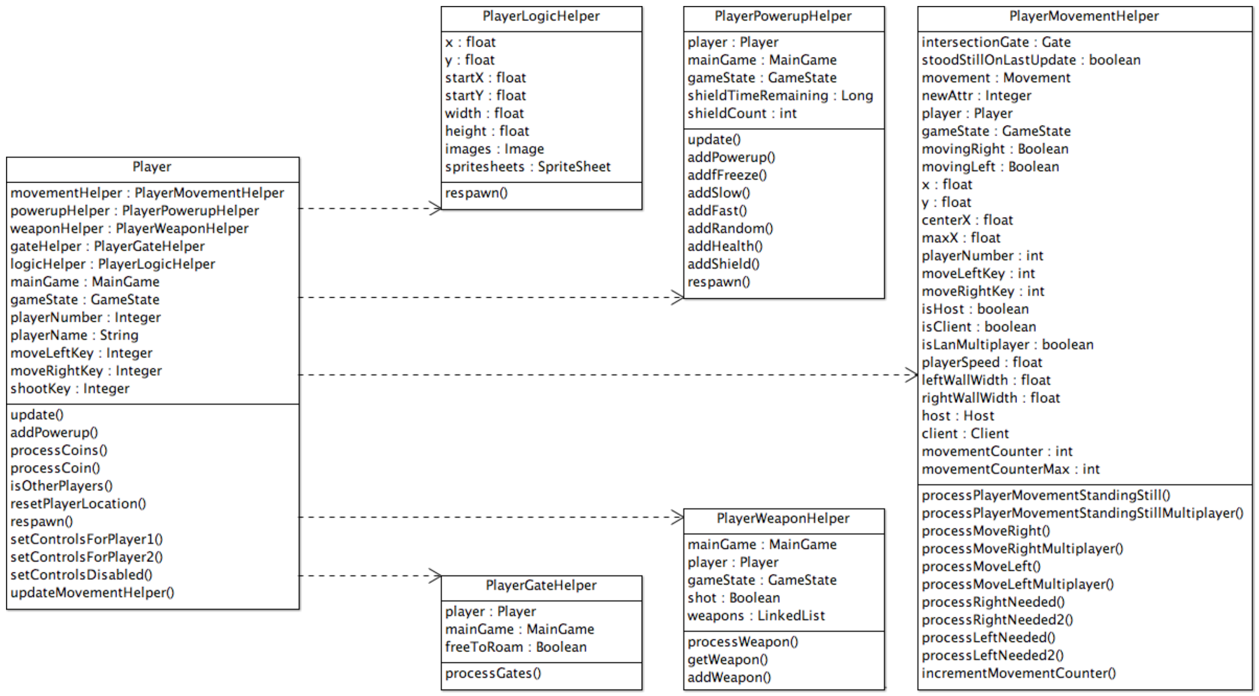
To fix this design flaw, we moved functionality to helper classes, used less getters and setters, and split up complex methods. In essence, this had the desired effect of moving towards the best practice of the Single Responsibility Principle.

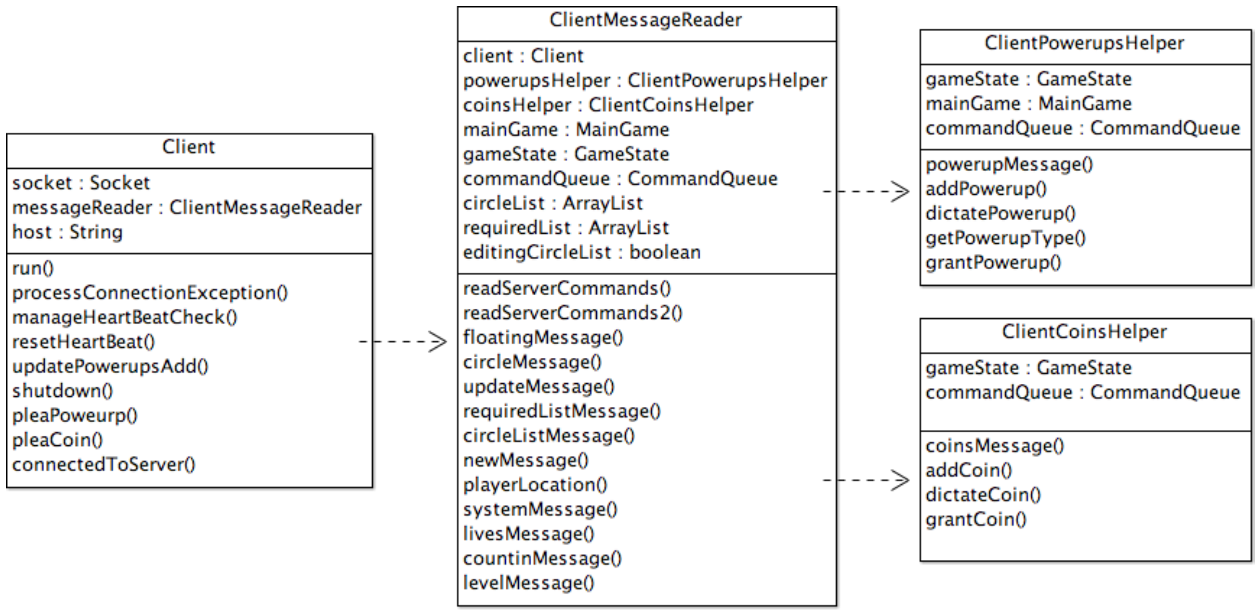
One of the god classes was Client, which, when it was first constructed, consisted of a way to connect to and communicate with a server for LAN multiplayer. This class later grew to encapsulate the reading and parsing of incoming messages from the server, thus making it responsible for nearly all client LAN functionality, resulting in its God Class status. This issue was addressed by keeping the reading/writing in Client and creating a few new classes for the parsing of the messages.

A similar approach was used for the Player and GameState classes, with the one difference being that the classes derived from the division of responsiblities were unique to the Player and GameState, which split the responsibilites among helper classes. In the case of GameState there was an abstract “helper” class which defined the protocol with which GameState could utilize the helper classes.

On the next few pages, we demonstrate the result by showing simple class diagrams for Player, GameState and Client. **These are also included separately, if necessary.**


GameStateHelper
mainGame : MainGame
parentState : GameState
enter()
exit()
update()
render()
GameStateCirclesHelper
circlelist : CircleList
shotList : ArrayList
processCircles()
updateActiveCircles()
updateShotCicles()
updateShotCircles2()
processUnlockCIrclesGates()
checkItem()
GameStateGateHelper
gateList : ArrayList
updateGateExistence()
GameStateInterfaceHelper
images : Image
newAttr : Integer
initImages()
renderBottomLayer()
renderTopLayer()
renderCountIn()
renderFloatingScores()
renderScore()
renderHealth()
renderShieldTimer()
renderShieldTimerPlayer1()
renderShieldTimerPlayer2()
GameStateItemsHelper
droppedPowerups : ArrayList
droppedCoins : ArrayList
speedPowerups : ArrayList
random : Random
processSpeedPowerups()
processCoins()
dropCoin()
dropPowerup()
renderPowerups()
renderCoins()
GameState
commandQueue : CommandQueue
circlesHelper : GameStateCirclesHelper
gateHelper : GameStateGateHelper
interfaceHelper : GameStateInterfaceHelper
itemsHelper : GameStateItemsHelper
levelsHelper : GameStateLevelsHelper
logicHelper : GameStateLogicHelper
pauseHelper : GameStatePauseHelper
playerHelper : GameStatePlayerHelper
enter()
exit()
exitOpacityZero()
exitOpacityNotZero()
init()
update()
updateOpacity()
render()
getID()
getCirclesHelper()
getGateHelper()
getInterfaceHelper()
getItemsHelper()
getLevelsHelper()
getLogicHelper()
getPauseHelper()
getPlayerHelper()
GameStateLevelsHelper
floor : MyRectangle
leftWall : MyRectangle
rightWall : MyRectangle
ceiling : MyRectangle
levels : LevelContainer
GameStateLogicHelper
totaltime : Integer
timedelta : long
timeremaining : long
prevTime : long
fractionTimeParts : Integer
countInStarted : boolean
countIn : boolean
playingState : boolean
levelStarted : boolean
waitForLevelEnd : boolean
updateCountIn()
switchCountInOff()
playGame()
processTime()
pauseStopped()
endLevel()
GameStatePauseHelper
returnButton : Button
menuButton : Button
exitButton : Button
waitEsc : boolean
pauseStarted()
newOperation()
GameStatePlayerHelper
weaponList : WeaponList


Player
movementHelper : PlayerMovementHelper
powerupHelper : PlayerPowerupHelper
weaponHelper : PlayerWeaponHelper
gateHelper : PlayerGateHelper
logicHelper : PlayerLogicHelper
mainGame : MainGame
gameState : GameState
playerNumber : Integer
playerName : String
moveLeftKey : Integer
moveRightKey : Integer
shootKey : Integer
update()
addPowerup()
processCoins()
processCoin()
isOtherPlayers()
resetPlayerLocation()
respawn()
setControlsForPlayer1()
setControlsForPlayer2()
setControlsDisabled()
updateMovementHelper()
PlayerLogicHelper
x : float
y : float
startX : float
startY : float
width : float
height : float
images : Image
spritesheets : SpriteSheet
respawn()
PlayerPowerupHelper
player : Player
mainGame : MainGame
gameState : GameState
shieldTimeRemaining : Long
shieldCount : int
update()
addPowerup()
addfFreeze()
addSlow()
addFast()
addRandom()
addHealth()
addShield()
respawn()
PlayerMovementHelper
intersectionGate : Gate
stoodStillOnLastUpdate : boolean
movement : Movement
newAttr : Integer
player : Player
gameState : GameState
movingRight : Boolean
movingLeft : Boolean
x : float
y : float
centerX : float
maxX : float
playerNumber : int
moveLeftKey : int
moveRightKey : int
isHost : boolean
isClient : boolean
isLanMultiplayer : boolean
playerSpeed : float
leftWallWidth : float
rightWallWidth : float
host : Host
client : Client
movementCounter : int
movementCounterMax : int
processPlayerMovementStandingStill()
processPlayerMovementStandingStillMultiplayer()
processMoveRight()
processMoveRightMultiplayer()
processMoveLeft()
processMoveLeftMultiplayer()
processRightNeeded()
processRightNeeded2()
processLeftNeeded()
processLeftNeeded2()
incrementMovementCounter()
PlayerWeaponHelper
mainGame : MainGame
player : Player
gameState : GameState
shot : Boolean
weapons : LinkedList
processWeapon()
getWeapon()
addWeapon()
PlayerGateHelper
player : Player
mainGame : MainGame
freeToRoam : Boolean
processGates()


Client
socket : Socket
messageReader : ClientMessageReader
host : String
run()
processConnectionException()
manageHeartBeatCheck()
resetHeartBeat()
updatePowerupsAdd()
shutdown()
pleaPoweurp()
pleaCoin()
connectedToServer()
ClientMessageReader
client : Client
powerupsHelper : ClientPowerupsHelper
coinsHelper : ClientCoinsHelper
mainGame : MainGame
gameState : GameState
commandQueue : CommandQueue
circleList : ArrayList
requiredList : ArrayList
editingCircleList : boolean
readServerCommands()
readServerCommands2()
floatingMessage()
circleMessage()
updateMessage()
requiredListMessage()
circleListMessage()
newMessage()
playerLocation()
systemMessage()
livesMessage()
countinMessage()
levelMessage()
ClientPowerupsHelper
gameState : GameState
mainGame : MainGame
commandQueue : CommandQueue
powerupMessage()
addPowerup()
dictatePowerup()
getPowerupType()
grantPowerup()
ClientCoinsHelper
gameState : GameState
commandQueue : CommandQueue
coinsMessage()
addCoin()
dictateCoin()
grantCoin()

2.2, Internal Duplications:

The PlayerTest class was reported as suffering from severe internal duplication. PlayerTest is a class we use to test the Player class. For testing the Player.update() method, in particular, we had to set up a lot of extra tests which shared a lot of duplicate code. This was done so we could test all possible branches of the Update() method quickly.

Instead of duplicating code for every test, we should have used parameterized tests, or more cleverly used the setup() method. Due to time-issues, however, we were not able to implement this nicely. We thus chose to sacrifice a marginal percentage of coverage instead. Incode now no longer marks PlayerTest as suffering from Internal Duplication.

2.2, Schizophrenic Class:

The RND class was reported as being a schizophrenic class. This was caused by RND's methods, which were static members, which in turn led to Incode reporting an extreme lack of cohesion.

When we first built this class, we made it static - and purely consist of draw functions - so that it would be easily accessible throughout our code. We did not assume instantiations would be necessary. This was a severe misjudgement on our side, leading to this particular design flaw.

To fix the flaw, we applied a singleton pattern to the RND class, effectively giving it the same capabilities as before, while massively increasing the cohesion. We also added a 'helper object' so that it could be better supplied variables.

All its methods are now non-static, and inCode no longer reports any schizophrenic classes.