D7032E
Home Exam

Author:

Marcus Asplund marasp-9@student.ltu.se

Institutionen för system- och rymdteknik

October 27, 2022

1 Unit testing

Things that do not work:
(1) There is no check that there are between 2 and 5 players
(3.a) The amount of defuses are not 2 for 2-4 players or 1 for 5 players but actually 6-PlayerAmount (example 4 defuses for 2 players)
(9.a) Player can write pass with lowercase and not draw card
(10.b) If you explode you still take extra turns if you have turns left
(11.a.i) Works except if you're attacked then the next player Pass the first turn and then attacks. you will get 4 turns not 2
(11.g) Did not work to type the command for playing three cards

The Program has other problems as well writing "pass" makes you Pass without drawing a card which is cheating

It is not possible to test the code since the constructor starts the game and the thread gets stuck waiting for player input. Was able to make one input with System.setIn() but was able to make any kind of testing with it. Also tried to use threads with a client and server but the game seemed to terminate anyways. accessing any variable when threading only gave me a null pointer (probably since the thread terminated).

2 Quality attributes, requirements and testability

The requirements stated are very vague. Since it is not explained what easy to test or easy to modify and extend means. To improve these requirements it should say how it should be easy to test and how it should be easy to modify the code. The consequences might be that it will not be as testable, modifiable or extendable as the stake holder wants the software to be. As there are no mentioned way this should be implemented the testing of the requirement is hard and somethings might be forgotten.

3 Software Architecture and code review

The code is very bad from a extensibility view point. The game loop is very complex. Dividing up the game loop to multiple primitive functions would make it more manageable. Adding any new cards would mean changing code in multiple places. or changes to gameplay would mean remaking a very large function. So to add something you would have to do it open-box. To improve extensibility you would have to divide up the code in more classes to make it and functions to with high cohesion and low coupling to make changes more easily. Adding more abstraction to handle all the cards would make it a lot easier to add new cards.

The code is also bad from a modifiability stand point. To gain better modifiability we should strive for high cohesion and low coupling but the code is the opposite with low cohesion and high coupling. Having Classes that are
Testability is very bad. To improve this we need primitive classes with low coupling so we can make unit tests. It is almost impossible since high cohesion of the classes. starting the program instantly demands input and the thread gets stuck.

4 Software Architecture design and refactoring

To improve DevCats code I decided to rewrite the whole project in C. To make it more comprehensive I divided it into multiple classes and functions. I tried my best to make these classes with primitive, with loose coupling and high cohesion in mind. This was to make it easier to understand the code but to also make it easier to test.

With cards and cardpacks i had extensibility in mind and tried to make it abstract to make it easy to add cards. by implementing a general overriding a abstract class of card and cardpacks.

I used the factory design pattern since i had the problem where i had to figure out which cardpack to load depending on a string. I found no abstract way to do this so the factory design pattern was my solution

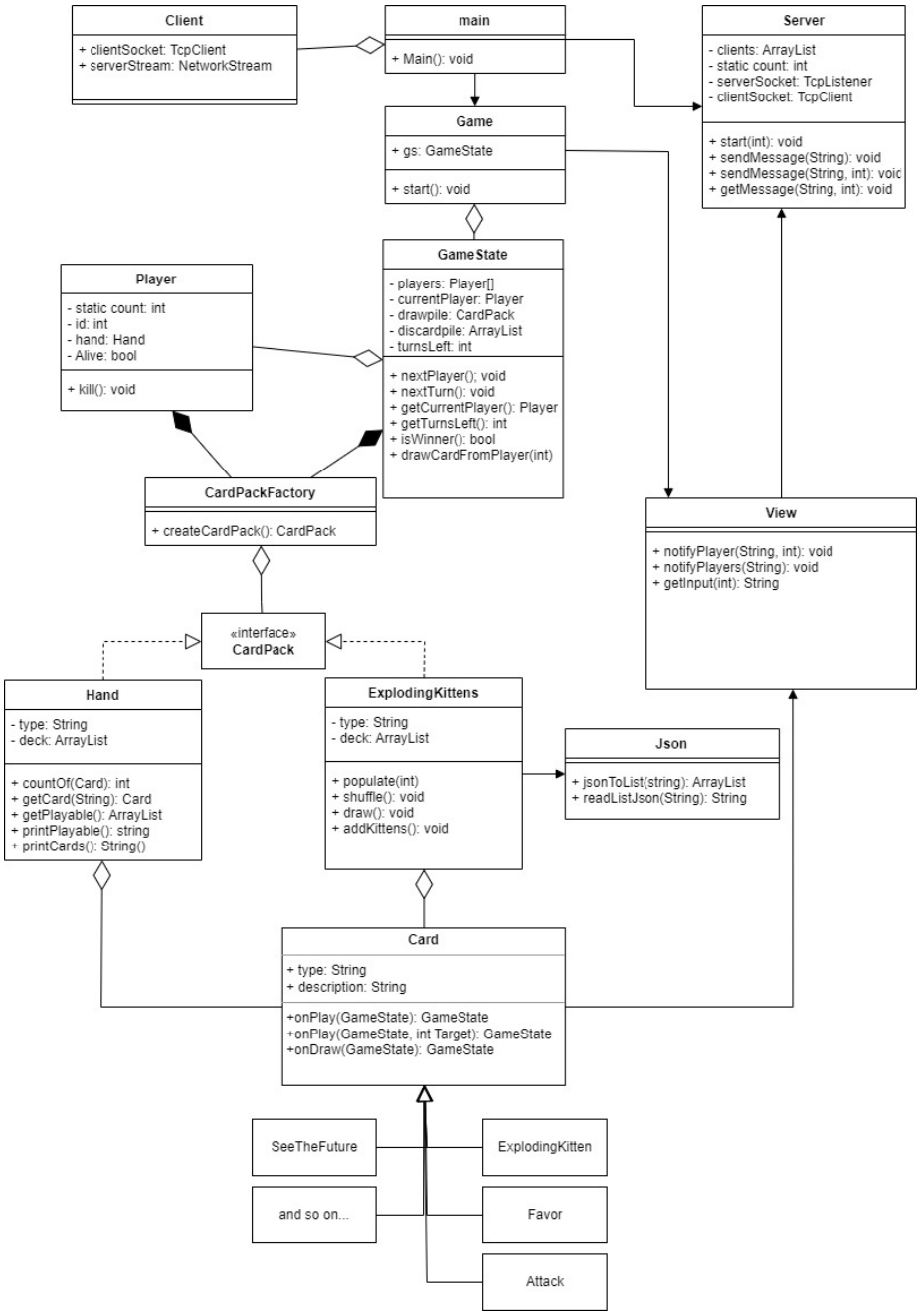
Client
+ clientSocket: TcpClient
+ serverStream: NetworkStream
main
+ Main(): void
Server
- clients: ArrayList
- static count: int
- serverSocket: TcpListener
- clientSocket: TcpClient
+ start(int): void
+ sendMessage(String): void
+ sendMessage(String, int): voic
+ getMessage(String, int): void
Game
+ gs: GameState
+ start(): void
GameState
- players: Player[]
- currentPlayer: Player
- drawpile: CardPack
- discardpile: ArrayList
- turnsLeft: int
+ nextPlayer(); void
+ nextTurn(): void
+ getCurrentPlayer(): Player
+ getTurnsLeft(): int
+ isWinner(): bool
+ drawCardFromPlayer(int)
Player
- static count: int
- id: int
- hand: Hand
- Alive: bool
+ kill(): void
CardPackFactory
+ createCardPack(): CardPack
View
+ notifyPlayer(String, int): void
+ notifyPlayers(String): void
+ getInput(int): String
«interface»
CardPack
Hand
- type: String
- deck: ArrayList
+ countOf(Card): int
+ getCard(String): Card
+ getPlayable(): ArrayList
+ printPlayable(): string
+ printCards(): String()
ExplodingKittens
- type: String
- deck: ArrayList
+ populate(int)
+ shuffle(): void
+ draw(): void
+ addKittens(): void
Json
+ jsonToList(string): ArrayList
+ readListJson(String): String
Card
+ type: String
+ description: String
+onPlay(GameState): GameState
+onPlay(GameState, int Target): GameState
+onDraw(GameState): GameState
SeeTheFuture
and so on...
ExplodingKitten
Favor
Attack

Figure 1: UML

References