

Programming Project Report

Business & IT

s2248719 Yin Liping(working alone)

Overall design	1
Class diagrams	1
Functional requirements overview	1
Implementation of MVC pattern	3
Testing	5
Unit testing	5
System testing	7
Reflection on planning	14

Overall design

Class diagrams

Explain the global design and the roles and responsibilities of each package and class. (Class diagram is not required)

packages	classes	Class responsibilities	Package responsibilities
abalone	Game	Implements all the basic functions for the game like move the marble, print the board, initialize a game board with marbles.	Create the model for the game including all the information about the game rules.
	Field	Another name for the marble on the board	
	Board	Place for the game to be played which is filled by well organized marbles by their position according to the game rules.	
	Marble	Marble with color, position and Boolean state as the basic elements of the game on the board.	
	Position	Position of the marble by (row, column)	
	Player	Abstract class to be extended by other player's type	
	ComputerPlayer	Computer player with strategy (not required to be implemented)	
	HumanPlayer	Human player with marbles and check the validation of the marbles by their positions; select marbles; check if the player is alive;	
	Strategy	Strategy for Computer player and also used as hint(not required to be implemented)	
protocol	Client	Ask players to enter IP and port number to create connection to the server; ask and deal with all kinds of information from player based on protocol and send to the server through ClientHandler class.	Implements the View and Controller of the game according the protocols and basic requirements of the server and the client.
	Server	Ask players to enter port number and create multithreaded synchronized games.	
	ClientHandler	Handle all the commands received from player(through client) to control the game process mainly by calling methods from game class. Send messages to the client as well.	
	GameEntity	Game entity of different number of players, in order to send the right messages to the right player.	
utils	Direction	Representatives of possible directions (based on protocol)which the marble can move to.	Creates some useful elements that can be reused by many classes
	MarbleColor	Representatives of marble color	
	ProtocolCode	Representatives of all protocol messages	
	TextIO	Input and output	
test	Game2Test	Test the function of class Game (some other classes and methods)	Unit test

Functional requirements overview

functional requirements of server

- 1.When the server is started, a port number should be entered :
 - Implemented by the `main()` method in Class Server.
- 2.Check if the port number has been used:
 - Implemented by method of `run()` in Class Server.

3.Multithreaded server to support multiple instances of the game being played simultaneously:

- Class server implements Runnable to be multithreaded and create new threads of CilentHandler in the *run()* method to realise multiple instances of games.

4.The server has a TUI that ensures that all communication messages are written to the console(System.out).

- Class server has CilentHandler instances in the *run()* method
- method of *handleCommand(String msg, Writer out)* and all the rest methods called by *run()* method in Class ClientHandler.

5.The server should respect the protocol:

- Class ProtocolCode includes all the possible messages between server and client , meanwhile methods mentioned above which realize the TUI function in Class CilentHandler are implemented according to the protocol.

functional requirements of Client

1.The client should have a user-friendly TUI that provides options to the user to request a game to the server.

- Constructor *public Client(String ip, String port)* and method *connect()* in class Client to ask the user to input valid IP and Port number by showing appropriate messages.

2.The client should support multiple human players per game (up to the number that the game rules require).

- *Main* method in class Client.

3.The client provides a hint functionality that shows a possible move to a human player, as calculated.After a game is finished, the player should be able to start a new game.

- Haven't been implemented yet(misunderstanding about the implementation of computer player and the hint, since the computer player is not required for working alone student).

4.The client should respect the protocol as defined for the pair of tutorial groups during the project session in Week 7, i.e., the client should be able to communicate with all the servers from the other pairs in the tutorial group.

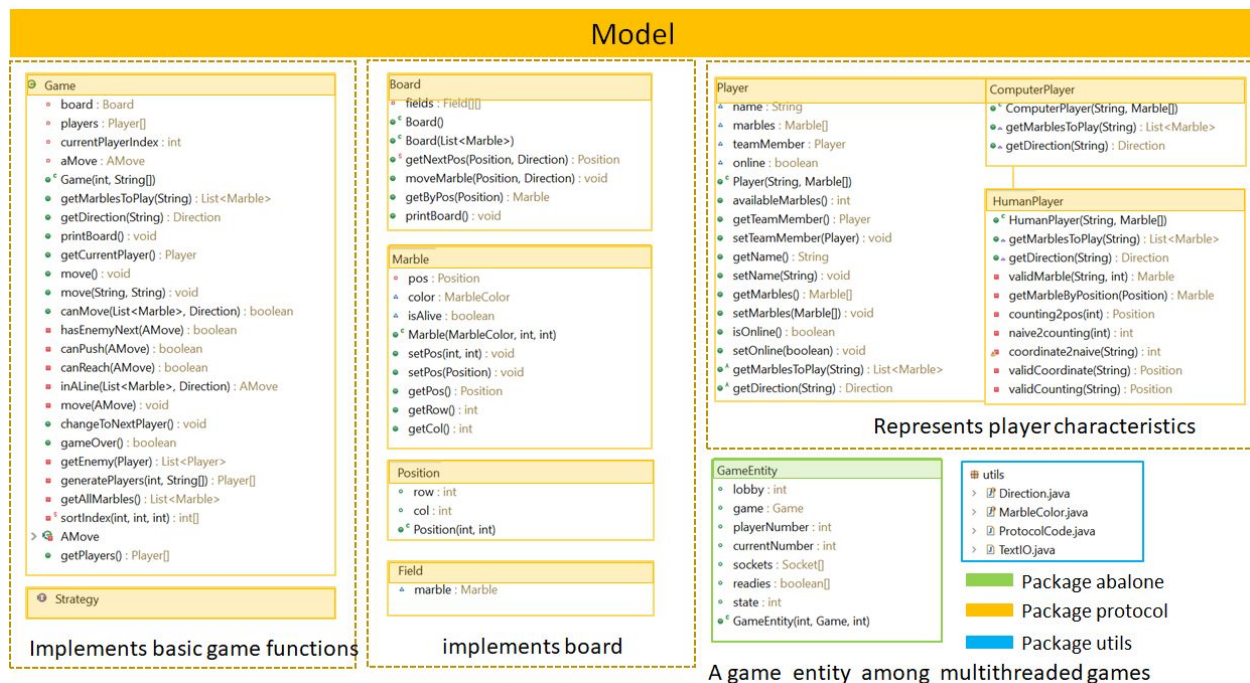
- Methods of *change(String msg)*, *sendToServer(String msg)*,*handleCommand(String msg)*,*connect()* and *main* method which calls these methods in Class Client.

5.problems about disconnecting,or players quit the game before it has finished, or the client crashes.

- Methods of *shutDown()*,*dealReconnect()*,*dealExit()*, *printToAllPlayer(GameEntity entity, String msg)* in class ClientHandler.
- Haven't been perfectly finished yet only part of it has been considered due to the time limit.

Implementation of MVC pattern

The models



Package abalone:

- Class Game with attributes of Board board, Player[] players and currentPlayer. It is constructed by number of players and player names. This class has all elements about this game.
- Class Board with 61 fields occupied by Marbles.
- Class Field with one attribute of Marble.
- Class Marble with Position, MarbleColor and Marble state.
- Class Position to represent the marble position on the board.
- Abstract Class Player, HumanPlayer and ComputerPlayer are classes which extend Player to represent who plays the game and which marbles belong to that player.

Package utils:

- enum Direction represents all the possible directions of the marbles which can make by the players during the game.
- enum MarbleColor represents the color of marbles.
- Class ProtocolCode represents all the possible commands sent between the server and client and some error messages sent to the user.

Package protocol:

- Class GameEntity represents the game state to a specific player.

View & Controller

```

Client
+ USAGE : String
+ STATE_ERROR : int
+ STATE_INITIAL : int
+ STATE_CONNECT : int
+ STATE_LOBBY : int
+ STATE_START : int
+ STATE_OVER : int
+ STATE_EXIT : int
+ ip : String
+ port : int
+ sock : Socket
+ out : BufferedWriter
+ in : BufferedReader
+ state : int
+ errorReceive : String
+ game : Game
+ player : Player
+ clientName : String
+ hasReceive : boolean
+ handleResult : int
+ Client(String, String)
+ main(String[]) : void
+ connect() : boolean
+ sendToServer(String) : boolean
+ createOrJoin() : boolean
+ change(String) : String
+ handleCommand(String) : int
+ dealError(String) : void
+ dealUpdate(String[]) : void
+ dealScore(String[]) : void
+ dealStart(String[]) : void
+ dealLobby(String[]) : void
+ printBoard() : void
    
```

1.Handles all the commands received from the users and send then to the server (through ClientHandler);
 2.Creates connection to the server;
 3.Implements TUI to communicate with user.

```

ClientHandler
+ gameEntities : List<GameEntity>
+ lobbyId : int
+ in : BufferedReader
+ out : BufferedWriter
+ sock : Socket
+ playerName : String
+ ClientHandler(Socket)
+ run() : void
+ handleCommand(String) : void
+ dealCreate(String, int) : void
+ dealJoin(String, int) : void
+ dealReady(String) : void
+ dealExit(String) : void
+ dealMove(String, String, String) : void
+ dealReconnect(String) : void
+ printToAllPlayer(GameEntity, String) : void
+ shutdown() : void
+ validName(String) : boolean
+ nameExists(String) : boolean
+ write(String) : void
    
```

1.Handles all the commands received from the users(through Client) and control the game process;
 2. Implements Runnable to play a game;
 3.Implements TUI to communicate with user.

```

Server
+ port : int
+ USAGE : String
+ Server(int)
+ run() : void
+ main(String[]) : void
    
```

1.Create multithreaded synchronized games;
 2. Ask user to input port number to listen to .

Package abalone

The controllers

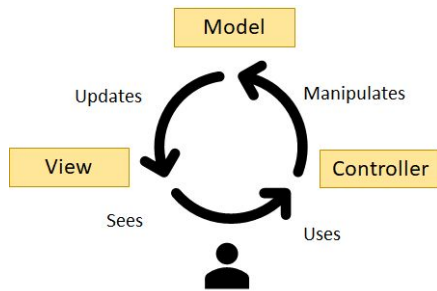
- Class ClientHandler: starts, proceeds and ends the game based on the commands received from the clients.
- Class Server : creates multithread game instances(controlled by ClientHandler).
- Class Client: set up the game and update the game state.

The views

- Class Client:
Receive and send back messages , at the same time check the validation of messages from the players and send commands to the server, meanwhile the server will check the validation of the commands.
- Class Server :
Asking the user the port number to create multithreaded games; Send the game updated state and other information about the game to players.
- Class ClientHandler
When an error occurs, this class outputs the message to the player.

Improvement strategies

Due to some reasons,the current version of MCV model is not well organized enough.The TUI of server and client hasn't been isolated out from the controller. To make a



more reasonable MCV model, the functions of these three components should be considered more carefully and organized well.

- The **Model** contains only the pure application data without considering the communication with the user.
- The **View** presents the model's data to the user and gets messages from the user.
- The **Controller** exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events.

Model

GameModel

Use the original game model which is shown above

Controller

ClientHandler

Rearrange the methods in original classes of Client, Server and ClientHandler; Keep the core functions of creating a Networking based game from the class Client and Server;

Client

ClientHandler are the main controller to handle all the messages sent by the view(the view as a messages transfer station, then get the user's messages and handle it according the protocol and then sent to the Server, and the ClientHandler controls the game process for the Server based on commands from users.)

Server

View

ClientTUI

Rearrange the methods in original classes of Client, Server and ClientHandler; All the functions related to sending and reading messages to the users should be implemented inside these two classes;

ServerTUI

Based on the protocols, make more specific methods can be made.

Testing

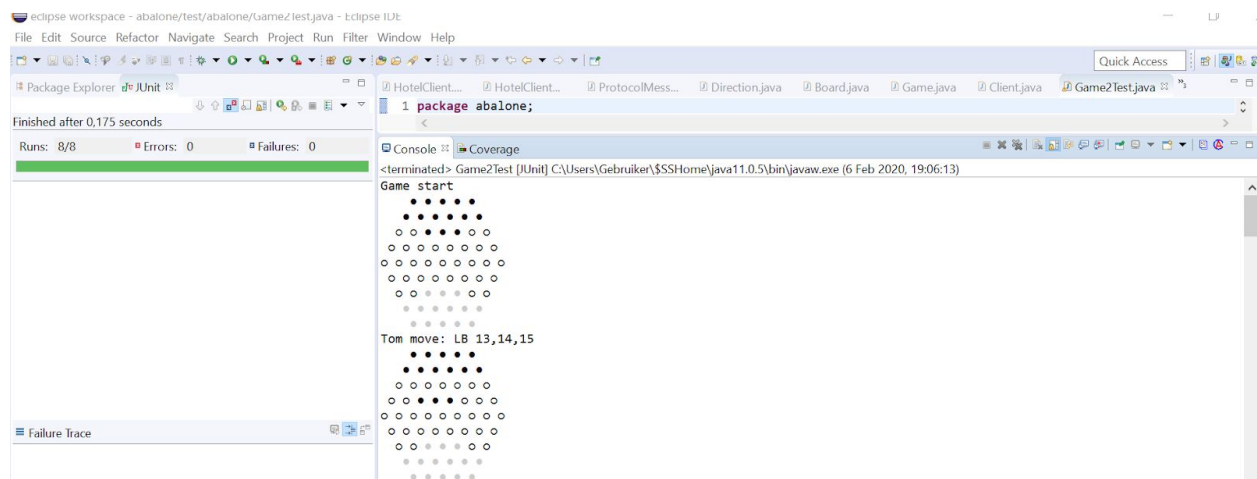
Unit testing

1. Test Strategy
 - together with other classes
2. Test method
 - Junit test with main method
3. Test result and expected results

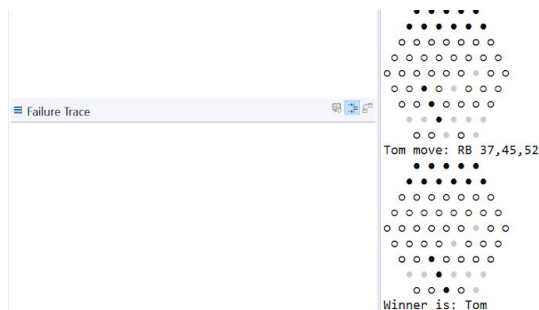
Test whether the game's function in class Game works well. Methods called inside the main method include: *printBoard ()*, *move (String dir, String marble)*, *changeToNextPlayer ()*, *getCurrentPlayer ()*, *gameOver ()* and *getName ()* from class *Player*.

Expected : the game can be played between two players successfully, players take turns and move the marbles.

Results : the game function works well.



```
eclipse workspace - abalone/test/abalone/Game2Test.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Filter Window Help
Package Explorer JUnit
1 package abalone;
Finished after 0.175 seconds
Runs: 8/8 Errors: 0 Failures: 0
Console Coverage
<terminated> Game2Test [JUnit] C:\Users\Gebruiker\SSSHome\java11.0.5\bin\javaw.exe (6 Feb 2020, 19:06:13)
Game start
  . . . . .
o o . . . . o
o o . . . . o
o o . . . . o
o o . . . . o
o o . . . . o
  . . . . .
Tom move: LB 13,14,15
  . . . . .
o o . . . . o
o o . . . . o
o o . . . . o
o o . . . . o
o o . . . . o
  . . . . .
```


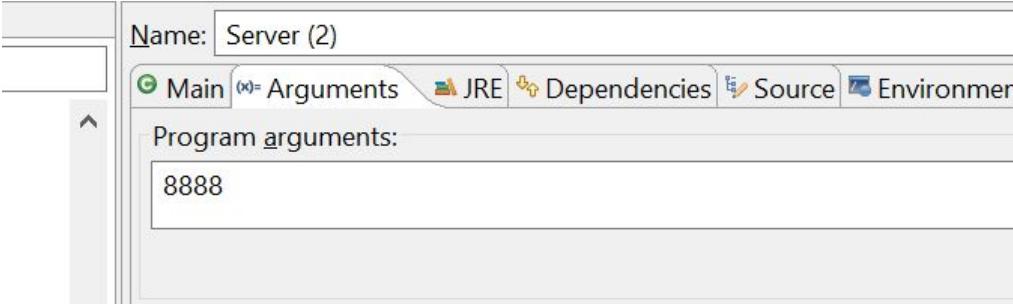


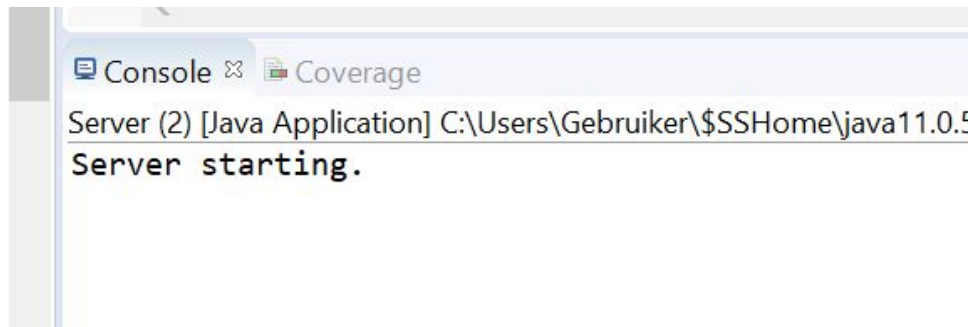
```
Failure Trace
  . . . . .
o o . . . . o
o o . . . . o
o o . . . . o
o o . . . . o
o o . . . . o
  . . . . .
Tom move: RB 37,45,52
  . . . . .
o o . . . . o
o o . . . . o
o o . . . . o
o o . . . . o
o o . . . . o
  . . . . .
Winner is: Tom
```


System testing

1. Main requirements of system

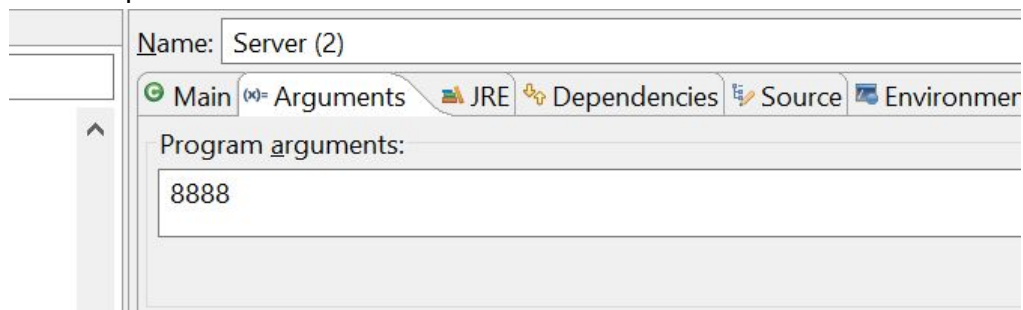
○ Test 1

Testing FR1
Functional requirement: Ask user to input a valid port number
Expected behavior: check if input is valid. If it is not, handle it properly: ask again or send warning messages.
Testing result: Test 1: input valid port number <ul style="list-style-type: none">- run:<p>The screenshot shows a code editor with a Java method signature <code>public void run() {</code> on line 17. Below the code, the console output is visible, showing the message <code><terminated> Server (2) [Java Application] C:\Users\Gebruiker\\$. Expected parameter: <port></code>.</p>- input:<p>The screenshot shows the 'Main' tab of an IDE. The 'Program arguments' field is set to '8888'. The 'Name' field is set to 'Server (2)'. The 'JRE' and 'Dependencies' tabs are also visible.</p>- output:success

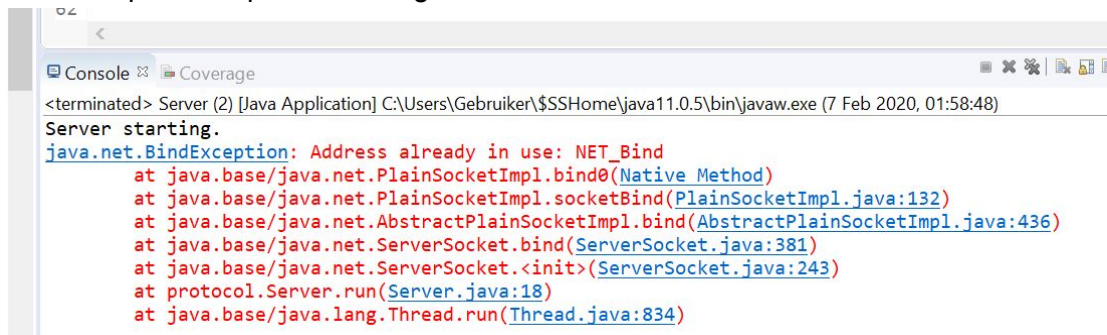


Test 2: input a used port number

- Input:



- Output: exception handling



Test 3: input a wrong format port number

- input:



- output: error warning and the player should enter new port number

```

<terminated> Server (2) [Java Application] C:\Users\Gebruiker\SSHome\java11.0.5\bin\javaw.
Expected parameter: <port>
ERROR: port BPPP is not an integer

```

- Test 2

Testing FR2

Functional requirement:

Players can only join a game which is not full and doesn't start, else the player should create a new game.

Expected behavior:

If a game is not full, the player can join successfully, otherwise the system asks the player to create a new game.

Testing result:

Test 1: join a new game

- Player 1 creates a new game

```

Client (1) [Java Application] C:\Users\Gebruiker\SSHome\java11.0.5\bin\javaw.exe (7 Feb 2020, 02:17:40)
Connect to Abalone game server successfully
Please Create(1) or Join(2) a abalone lobby: 1
Please enter your name and player numbers(2-4) in the lobby(eg. Tom 4): Tom 2
This is a 2 players lobby: Tom. If you are ready to play, please send 'READY'

```

ready

- Player 2 joins

- Result: success!

```

Client (1) [Java Application] C:\Users\Gebruiker\SSHome\java11.0.5\bin\javaw.exe (7 Feb 2020, 02:17:40)
Connect to Abalone game server successfully
Please Create(1) or Join(2) a abalone lobby: 2
Please enter your name and player numbers(2-4) in the lobby(eg. Tom 4): Jack 2
This is a 2 players lobby: Tom,Jack. If you are ready to play, please send 'READY'

```

ready

Game start. Players are: Tom,Jack,

```

  . . . . .
 . . . . .
o o . . . o o
o o o o o o o
o o o o o o o
  o o . . . o o
  . . . . .
  . . . . .
Wait for Tom move...

```

Test 2: join a game which is full or has already started

- Player 3 joins the game which the two players above have already started and is

full.

- Result: fail ! and the system ask the player to create a new game

```
Client (1) [Java Application] C:\Users\Gebruiker\SSHome\java11.0.5\bin\javaw.exe (7 Feb 2020, 02:20:51)
Connect to Abalone game server successfully
Please Create(1) or Join(2) a abalone lobby: 2
Please enter your name and player numbers(2-4) in the lobby(eg. Tom 4): Tom 2
Player name already taken.
Please enter your name and player numbers(2-4) in the lobby(eg. Tom 4): Susan 2
No lobby, please create one
Please Create(1) or Join(2) a abalone lobby:
```

-

- o Test 3

Testing FR3

Functional requirement:

The Client should support multiple players to play the game simultaneously.

Expected behavior:

Multiple players can play the game according the protocols

Testing result:

Test 1: test two players

- Starts a game
- Results: successful !
- Plays the game:
- Successful!
- Player 1 plays(the process has been omitted since the picture is too long):

MOVE;LB;13,14,15

— — —

```
MOVE;RB;37,45,52
```

Game over. Winner is: Tom

—

—

—


```

145      }

```

Console Coverage

Server (2) [Java Application] C:\Users\Gebruiker\SSHome\java11.0.5\bin\javaw.exe (6 Feb 2020, 18:35:48)

```

Server starting.
Client connected!
CREATE;Tom;2
READY;Tom
Client connected!
JOIN;Jack;2
READY;Jack
MOVE;Tom;lb;13,14,15
MOVE;Jack;lt;45,46,47
MOVE;Tom;rb;20,21,22
MOVE;Jack;lt;37
MOVE;Tom;lm;29,30,31
MOVE;Jack;rm;38,39
MOVE;Tom;lb;28,29,30
MOVE;Jack;lb;27
MOVE;Tom;lm;36,37,38
MOVE;Jack;rt;55,60
MOVE;Tom;rt;35
MOVE;Jack;lb;49,55
MOVE;Tom;rt;27,36
MOVE;Jack;rt;55,60
MOVE;Tom;rb;19,28,37
MOVE;Jack;lb;49,55
MOVE;Tom;rb;45,28,37
MOVE;Jack;rt;55,60
MOVE;Tom;rb;52,45
MOVE;Jack;lb;49,55
MOVE;Tom;lt;52,58
MOVE;Jack;lm;59,60
MOVE;Tom;rb;37,45,52
MOVE;Jack;rm;59

```

2. solved Bugs examples

-If one player wants to join a lobby which hasn't been created, the system handles this properly. eg. let the player create a new lobby.(by calling *dealErr()* in method of *handleCommand()* in class client)

```

Connect to Abalone game server successfully
Please Create(1) or Join(2) a abalone lobby: 2
Please enter your name and player numbers(2-4) in the lobby(eg. Tom 4): James 2
Player name already taken.
Please enter your name and player numbers(2-4) in the lobby(eg. Tom 4): Kevin
Please enter your name and player numbers(2-4) in the lobby(eg. Tom 4): Kevin 2
No lobby, please create one
Please Create(1) or Join(2) a abalone lobby:

```

3. Unsolved Bugs examples

- If a player disconnects , the application can't give proper response. His opponent will keep waiting without doing anything .

-

```
Wait for Tom move...
Player Tom move.
  ● ● ● ● ●
 ● ● ● ● ●
○ ○ ○ ○ ○ ○
○ ○ ○ ○ ○ ○
○ ○ ● ● ● ○ ○
○ ○ ● ● ● ○ ○
  ○ ○ ○ ○ ○
   ○ ○ ○ ○ ○
    ● ● ● ● ●
     ● ● ● ● ●

You turn. Please enter direction and marbles to move(e.g. MOVE;LB;7,13):
MOVE;LT,37
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 2 out of bounds for length 2
    at protocol.Client.change(Client.java:196)
    at protocol.Client.main(Client.java:80)
```

Reflection on planning

1. How was your planning influenced by your experiences with the planning and time writing during the Design project?

- What influenced me most is knowing how to arrange the time properly based on the importance level of the tasks and how to finish a hard and big project step by step by dividing them into easier and smaller issues.
- During the process, I realized how to identify the procrastinations and how to avoid them and how to motivate myself when things get harder and even beyond my ability. These skills help me manage time better and deal with the project more efficiently and effectively.

2.To what extent did your planning correspond to the actual progress during the project weeks? What made you deviate from your planning? For example:Were there project tasks that took significantly more or less work than you had expected? If so,why did this happen?Were there significant losses of time or momentum in your projects (one or more periods in which you just did not seem to make the progress you had intended)? Looking back, how do you explain this?

- I can say that only 50% correspondence to the progress of this project. The most significant reason is that I didn't expect the project would be so hard when doing it by myself. However I have no choice since my partner is not motivated at all and when I want to find a new partner to do this I found that it is too late since I don't want to take a risk of doing this project with a person I don't know any more.
- When big problems occur the progress can be hampered much for days, and I should take the responsibility of not asking others' help actively. More over this project make me realize more about the programming that a little bug can destroy the whole thing if I can't find out the reason.

3. Which countermeasures did you take to compensate for deviations from your original planning? What was the impact of this on the intended scope or quality of the project?

- When I have deviations from the project, I will work longer next day on the project to catch up with my schedule. However the efficiency could be still low and sometimes it couldn't guarantee a good quality of the project. As for the scope of the project, it could be obviously affected by the deviations since when I stop doing something it means no progress and even if I try very hard to catch up but I was still behind the schedule.
- I think the reason is not hard to find. If it is because the difficulty goes far beyond the limits of my abilities, then the more time I waste the less chances I could solve the problem. If it is because of procrastination then I deserve the bad result of the project.

4. What did you learn from this experience for your next (project) planning? Take your answers to the other questions into account and ask yourself how you would want to prevent this or deal with this next time.

- The best way is to plan it as early as possible and ask advice from TAs or anyone who knows more than I about the project. This can avoid many problems when I actually do the project.
- When my programming ability is not enough for the project and there is something that I can't solve, I should ask for help from TAs in time. Don't let any reason to stop the progress of the project or when the deadline approaches things can be even tougher.

5. Suppose that next year you are a teaching assistant for this project. Give at least two do's and don'ts that you would tell your students to help them with their planning.

- Make sure to test your classes and methods at the moment they are finished, since it is quite difficult to test them all in the end.
- Making a good overall design from the beginning helps a lot to make the test and modification easier afterwards.
- Try your best to avoid procrastination, since the project is a lot of work.
- A good partner helps you a lot, at least find a motivated partner if his new learner of programming.