5 April 2020

Software Engineering II

Version 1.2.3

Game Documentation

Elif Ozdemir
Assyl Salah
Nikita Zakharov
Vladislav Sorokin
Denis Harbatsenka

Instructors: Aleksander Kosicki, Michal Okulewicz

Contents

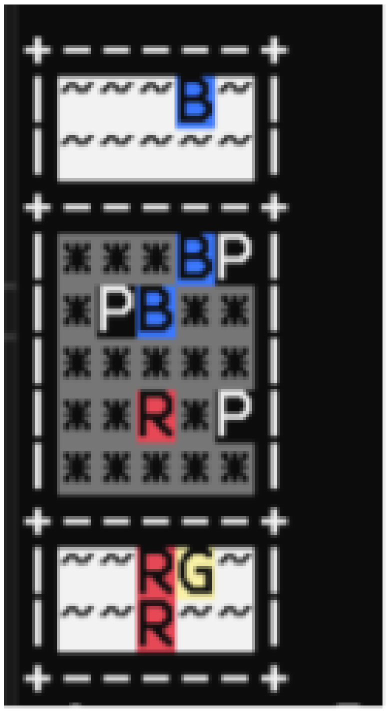
1 Changelog

- 1.1.0 2020-03-10 Updated all the changes occurred in issue:
  1, 6, 7, 8, 10, 14, 15, 16, 17.
- 1.1.1 2020-03-11 Removed communication between players (updated the use cases diagrams).
- 1.1.2 2020-03-16 Updated class diagram and updated all the changes occurred in issues:
  10, 11, 18, 19, 20, 21, 22, 25, 26, 27, 28, 29.
- 1.1.3 2020-03-16 Removed communication between players (in game interaction section), made changes in (update players info activity diagram) and updated all the changes occurred in issues:
  2, 4, 5, 23, 24.
- 1.1.4 2020-03-16 Updated class diagram and updated all the changes occurred in issues:
  31, 32.
- 1.1.5 2020-03-19 Changes in sequence diagrams and messages from XML to JSON.
- 1.1.6 2020-03-19 Updated all the changes occurred in issue:
  33, 34.
- 1.1.7 2020-03-24 Updated all the changes occurred in issue:
  36.
- 1.1.8 2020-03-30 Fixed some typos, added my team board version and updated all the changes occurred in issue:
  3, 37, 39.
- 1.1.9 2020-03-30 New class diagram, new feature player have GUI.
- 1.2.0 2020-04-01 More information about gui and updated all the changes occurred in issue: 41.
- 1.2.1 2020-04-02 Changed two methods in player class
- **1.2.2 2020-04-04 New class diagram (changes are mentioned above the class diagram).**
- **1.2.2 2020-04-05 New class diagram (changes are mentioned above the class diagram).**

2 Introduction

The system organizes matches between 2 teams of 4 players. The game is played in real time on a board, consisting of the tasks area and the goals area. Each team of players needs to complete an identical task, consisting of placing a set of pieces, picked from the tasks area, in the project goals area. The pieces appears at random within the tasks area. The player's view of the tasks area is limited and the shape of the goals needs to be discovered. The game is won by the team of players which is first to complete the project.The architecture of a system and a communication protocol design should focus on solving the effects of asynchronous actions taken by the players.

2.1 Example of how our board look like



```
+-----+
|~~~B~|
|~~~~~|
+-----+
|***BP|
|*PB**|
|*****|
|**R*P|
|*****|
+-----+
|~~RG~|
|~~R~~|
+-----+
```

- B:blue player.
- R:red player.
- *: unknown task field.
- Telda: unknown goal area.
- G: we placed piece into goal area and it was a goal field.
- +, |, - : border indicators.

2.2 Description of Figure 1

1. Red goal area can not touch Blue goals area.

2. All areas have rectangle shape.

3. Task area can be rectangular.

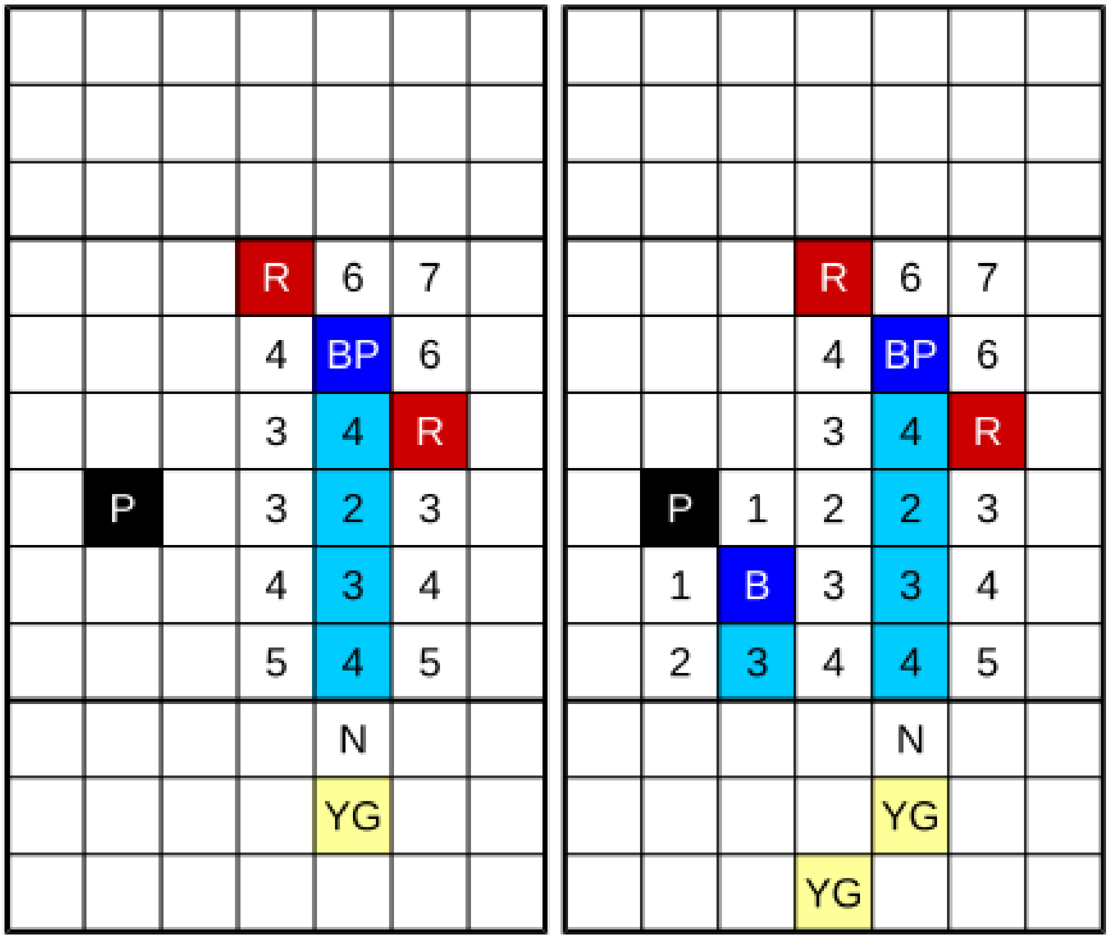
4.Width of all areas are the same and height of goal area is the same for both team.

5. Size of goals area and number of goals to discover are identical for both teams.

6. Goals relative position in goals are not identical for both teams.

7. Console background color is white.

8. Console foreground color is black.



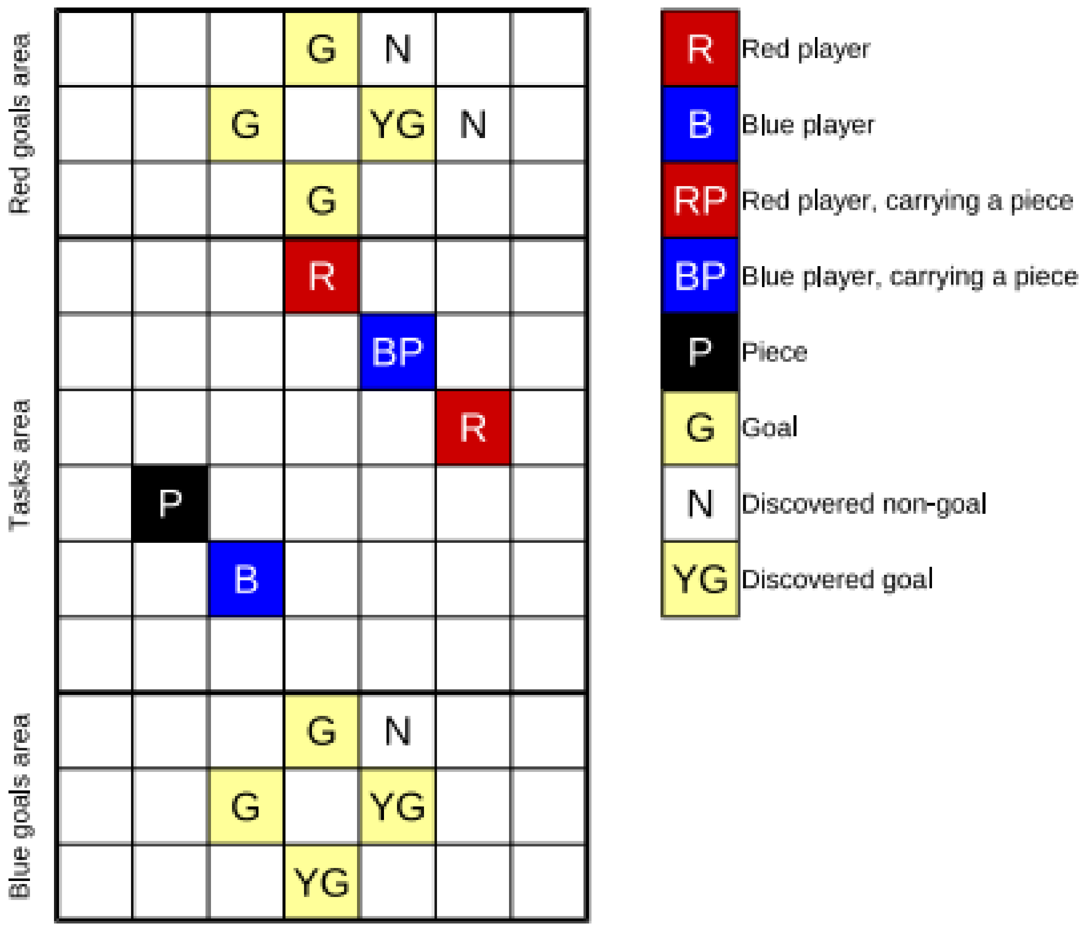


Figure 1: Global state of the game.

3 System requirements

3.1 User stories

As a host I could create lobby so that players could join to it.
As a host I could configure game properties so that we would enjoy game more.
As a team we can have check that everyone is ready so that we could start the game.
As a player I could check a piece so that I would know what kind of piece it is.
As a host I could configure where goals are so that team could train before the real game.

3.2 FURPS

Functionalities

Game Setup:

1. The game system allows to play the game.

2. The game should be played on a computer.

3. The players should be able to join the game before the game starts.

4. It's possible to play one game at a time i.e. CS and GM can only serve one game at the same time and player can use his console to play only one game (but he can open more than 1 to play multiple games simultaneously).

5. Game Master and Player are the main components of the system.

6. The game consists of two teams and a team consists of 1-4 players.

7. One of the players is the host.

8. All the communications between Game Master and Player should be handled by the server.

9. The communication server handles everything at the lower level without processing the content.

10. Game Master should be aware of and able to process all requests sent to him from Player.

11. Game Master should make all the decisions locally according to the requests made to it.

12.Each component of the game such as : game master, communication server don't have GUI.

13.Players have GUI.

14.Each player/communication server/game master has their own log file (No synchronization between them).

15.TCP/IP connection technology is requested between (Player and Communication Server) and between (Communication Server and Game Master).

16.Game Master connects to Communication Server

17.Only human players are allowed to play but for Game deadline it is allowed to use bots.

Game Play:

18. The rules that are valid during the game play are described in details in Game rules section of the documentation.

Usability

1. The full documentation is provided.
2. A manual for the user is included in the documentation.
3. Some use-case diagrams will be provided. 4. The game is playable on one computer and via network.

Reliability

1. Personal data is protected.
2. In case of a failure, user should be provided with an error message describing what exactly went wrong.
3. If there were 5 unsuccessful attempts to send message from server, the player is disconnected.

4.When game crashes its needs to be started from beginning.

Performance

1. All team members are supported at the same time.
2. The maximum response time is 1 second.
3. The memory consumption is low

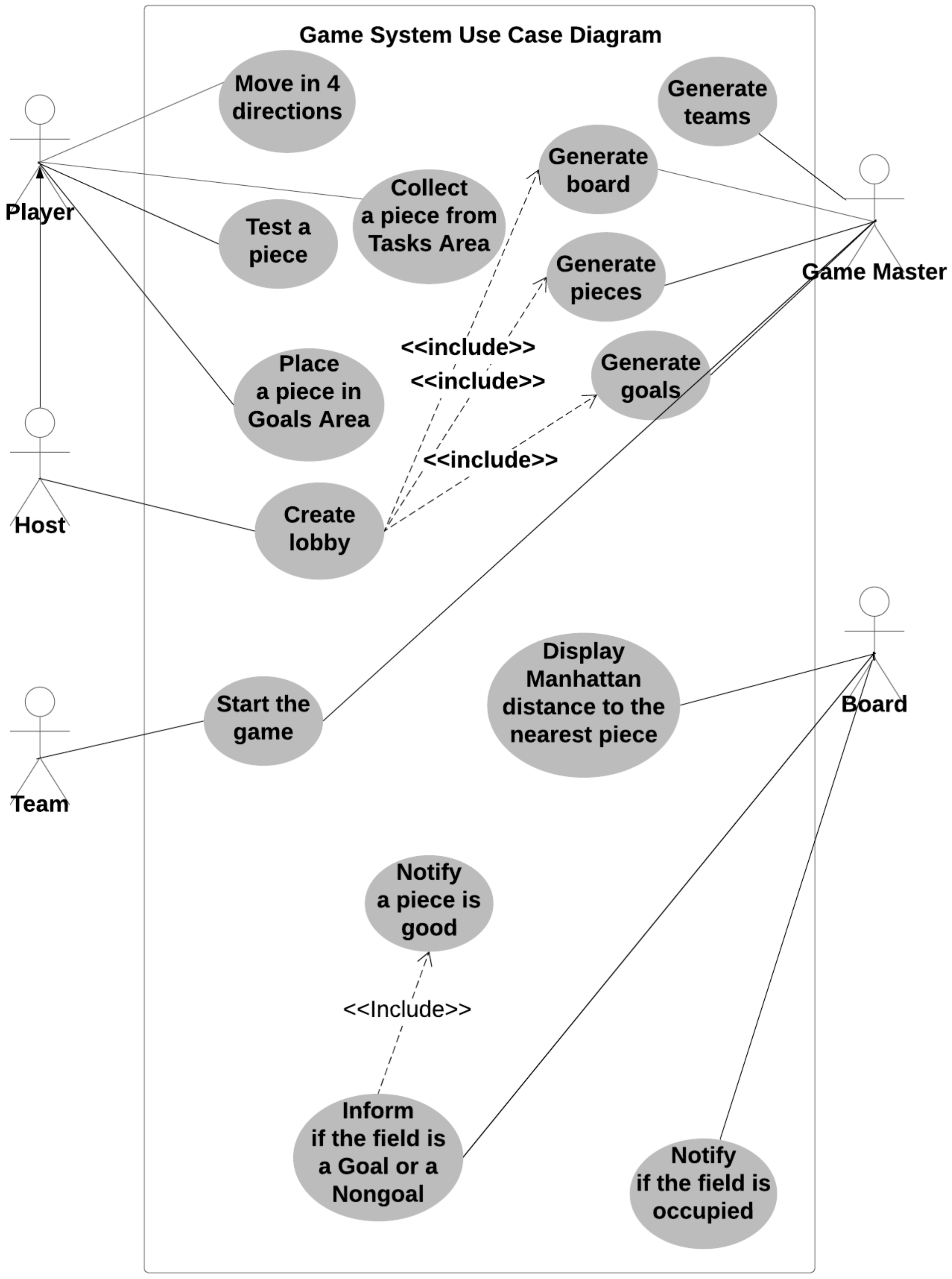
Supportability

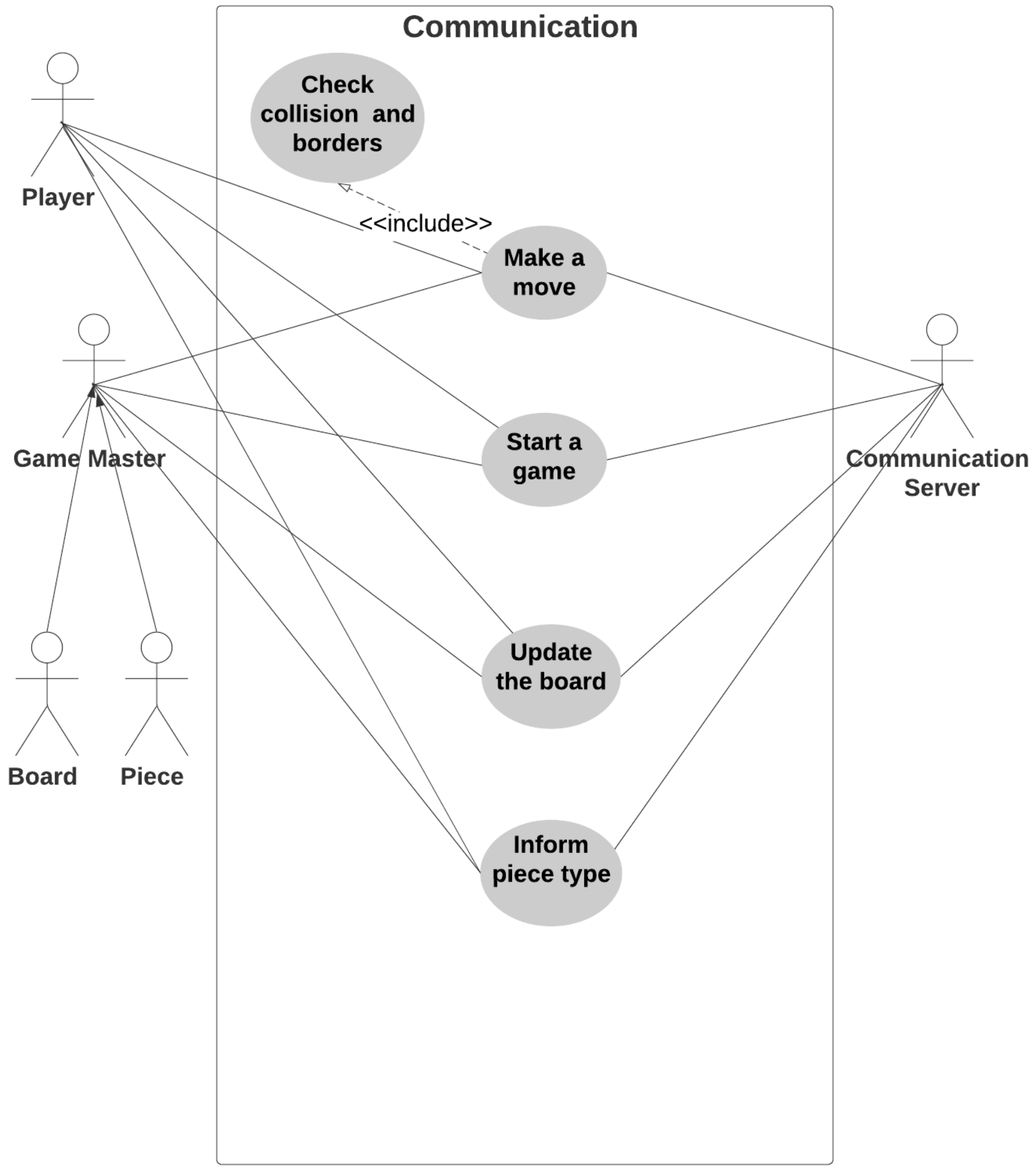
1. The game is installable.
2. System should support Windows.

3.Players , communication server and game master should run locally on one lab computer.

4.Both communication server and game master run only locally.

5. To understand the role of each user in the game, use-case diagrams should be provided.


Game System Use Case Diagram
Player
Host
Team
Game Master
Board
Move in 4 directions
Collect a piece from Tasks Area
Test a piece
Place a piece in Goals Area
Create lobby
Start the game
Generate teams
Generate board
Generate pieces
Generate goals
<<include>>
<<include>>
<<include>>
Display Manhattan distance to the nearest piece
Notify a piece is good
<<Include>>
Inform if the field is a Goal or a Nongoal
Notify if the field is occupied

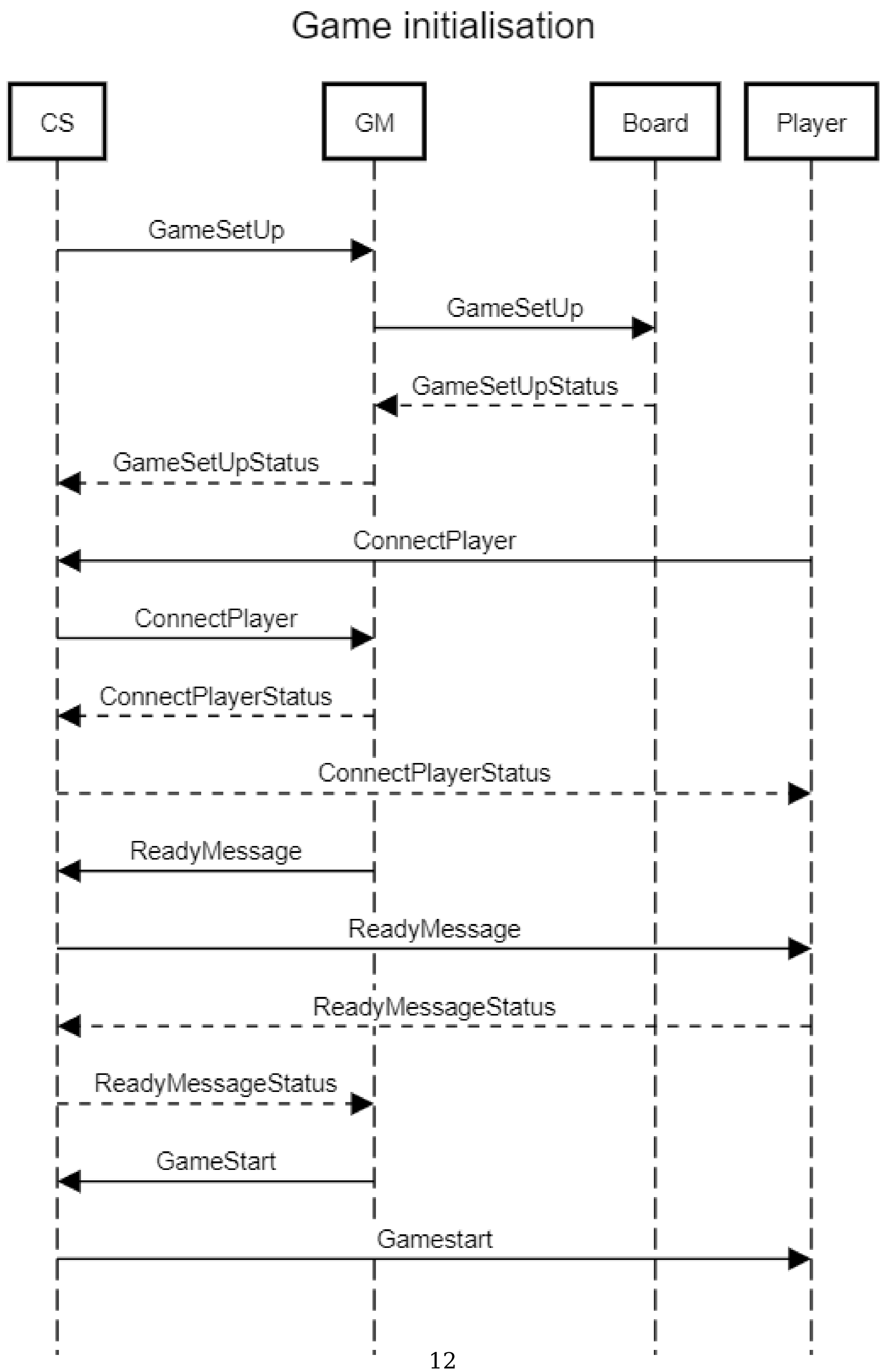

Communication
Check collision and borders
<<include>>
Make a move
Start a game
Update the board
Inform piece type
Player
Game Master
Board
Piece
Communication Server

4 Design Documentation

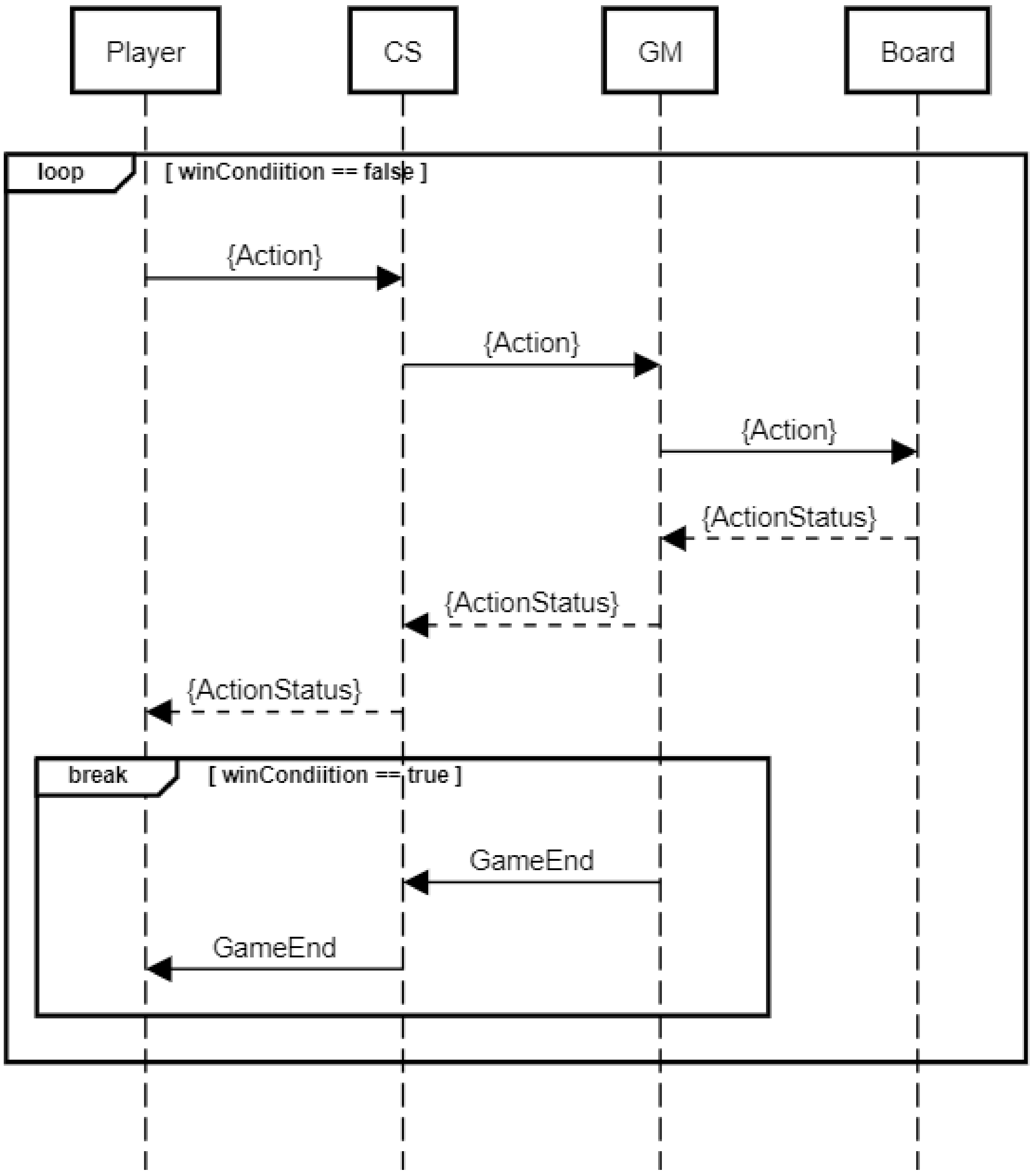
4.1 Architecture design

The system consists of three components which are Player, Game Master and Communications Server. Player is an agent playing the game, holds its own view of the state of the game. Game Master generates the board and the shape of the project goals, holds the real state of the game, generates new pieces on the tasks area and generates teams. The game stays under control in real time by the Game Master on regular basis till the end of the game.Game Master should queue received data (by receive timestamp), and act on them in order of receiving (FIFO). Buffer for incoming messages = 1024 bytes.
Communications Server is responsible for passing messages between Game Master and Players whenever some action is taken by the player like movement or taking a piece. Communication server is sending the message to the Game Master to handle the actions and to send a signal back to confirm the action and update the board. Game Master can communicate with players to restrict them from doing forbidden moves, giving them permission only if the action is valid.

4.2 Communication protocol design



Game Actions



Type of messages

- Connection message
- Status message
- Communication message
- Setup message message
- Action message
- State message
- Control message
- Update message
- End game message
- Statistic message

Some delays

- Move delay 100ms
- Discovery delay 500ms
- Test delay 1000ms
- Destroy piece delay 2950ms
- Place next piece delay 3000ms
- Pick-up delay 100ms
- Placing delay 100ms
- Update fail delay (if update failed for someone delay till next update) 1000ms

Game set up messages and respones

```
"GameSetup message"
{
  "action": "setup",
}

"GameSetup status message"
{
  "action": "setup",
  "status": "<OK|DENIED>"
}

"Connect player message"
{
  "action": "connect",
  "portNumber": "2020",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e",
}

"Connect player status message"
{
  "action": "connect",
  "portNumber": "2020",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e",
  "status": "<OK|DENIED>"
}

"Ready message"
{
  "action": "ready",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e"
}

"Ready status message"
{
  "action": "ready",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e"
  "status": "<YES|NO>"
}

"Game start message"
{
  "action": "start",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e",
  "team": "<Red|Blue>",
  "teamRole": "<Leader|Member>",
  "teamSize": <1|2|3|4>,
  "teamGuids": [
      "0f8fad5b-d9cb-469f-a165-70867728950e",
      "128fad5b-ddg4-4f9f-1235-7fg86h728950e",
      ...
    ]
  "position": {
    "x": <0|1|2|...|boardWidth in the config JSON - 1 >
    "y": <0|1|2|...|boardHeight in the config JSON - 1 >
  }
  "board":{
    "boardWidth": <1|2|3|...|boardWidth in the config JSON>,
    "taskAreaHeight": <1|2|3|...|taskAreaHeight in the config JSON
       >,
    "goalAreaHeight": <1|2|3|...|boardWidth in the config JSON>
  }
  }
  "GameEnd message"
{
  "action": "end",
  "result": "<Red|Blue>",
}
```

Game action messages and responses

```
"Move message"
{
  "action": "move",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e",
  "direction": <"Up|Down|Left|Right">
}

"MoveStatus message"
"if 'status' is DENIED, the 'position' is null"
{
  "action": "move",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e",
  "direction": "<Up|Down|Left|Right>",
   "status": "<OK|DENIED>",
  "position": {
    "x": <null|0|1|...|boardWidth in the config JSON - 1 >
    "y": <null|0|1|...|boardHeight in the config JSON - 1 >
  }

}

"PickUp message"
{
  "action": "pickup",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e",
}

"PickUpStatus  message"
{
  "action": "pickup",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e",
  "status": <OK|DENIED>
}

"Test message"
{
  "action": "test",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e"
}

"TestStatus  message"
"if 'status' is DENIED, the 'test' is null"
{
  "action": "test",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e",
  "test": <true|false|null>,
  "status": <OK|DENIED>
}

"Place message"
{
  "action": "test",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e"
}

"PlaceStatus message"
"if 'status' is DENIED, the 'placementResult' is null"
{
  "action": "test",
  "playerGuid": "0f8fad5b-d9cb-469f-a165-70867728950e",
  "placementResult": "<Correct|Pointless>",
  "status": "<OK|DENIED>"
}


"Discover messages"
"goalAreaHeight and taskAreaHeight are used to specify the
   dicvover bounds as it should work on task area only. These
   values will be taken from the config JSON"
{
  "action": "discover",
  "playerGuid": "0f8fad5b—d9cb—469f—a165—70867728950e",
"position": {
"x": <0|1|...|boardWidth in the config JSON – 1 >
"y": <goalAreaHeight|goalAreaHeight + 1|...|goalAreaHeight +
   taskAreaHeight
in the config JSON –1>
  }
}
"DiscoverStatus messages"
"goalAreaHeight and taskAreaHeight are used to specify the
   dicvover bounds as it should work on task area only. These
   values will be taken from the config JSON"
"distance == manhattanDistance"
"'field' may contain up to 9 elements"
"if 'status' is DENIED, then 'fields' is empty"

{
"action": "discover",
"playerGuid": "0f8fad5b—d9cb—469f—a165—70867728950e",
"position": {
"x": <0|1|...|boardWidth in the config JSON –1>,
"y": <goalAreaHeight|goalAreaHeight
+1|...|goalAreaHeight +
taskAreaHeight
in the config JSON – 1>"
  },
"fields": [
       {
"x": <0|1|...|boardWidth in the  configJSON –1 >,
"y": <goalAreaHeight|goalAreaHeight + 1|...|goalAreaHeight +
   taskAreaHeight
in the config JSON –1>,
        "cell": {
    "cellState": "<Empty|Piece>"
"distance": <null|0|1|...|Nearest piece distance>,
 "playerGuid": "0f82adsb—dgcb—sf9f—ad65—70867fgjf950e"
        },
        {
"x": <0|1|...|boardWidth in the config JSON –1 >,
"y": <goalAreaHeight|goalAreaHeight + 1|...|goalAreaHeight +
   taskAreaHeight
in the config JSON -1>,
"cell": {
"cellState": "<Empty|Piece>"
"distance": <null|0|1|...|Nearest piece distance>,
"playerGuid": null
        }
        ...
      }
    ]
  "status": "<OK|DENIED>",
}
```

4.3 Special system states description

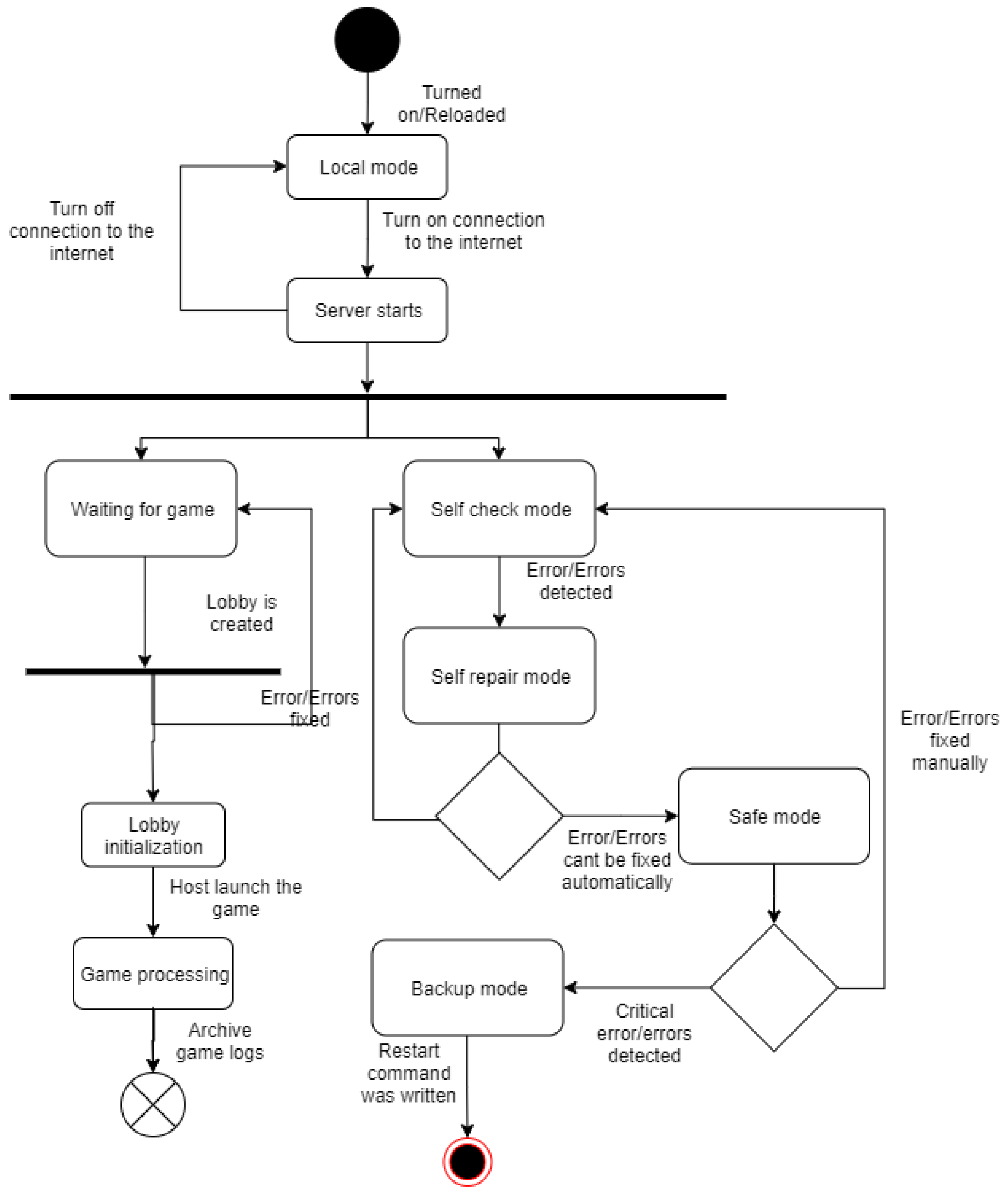
Explanation of server modes

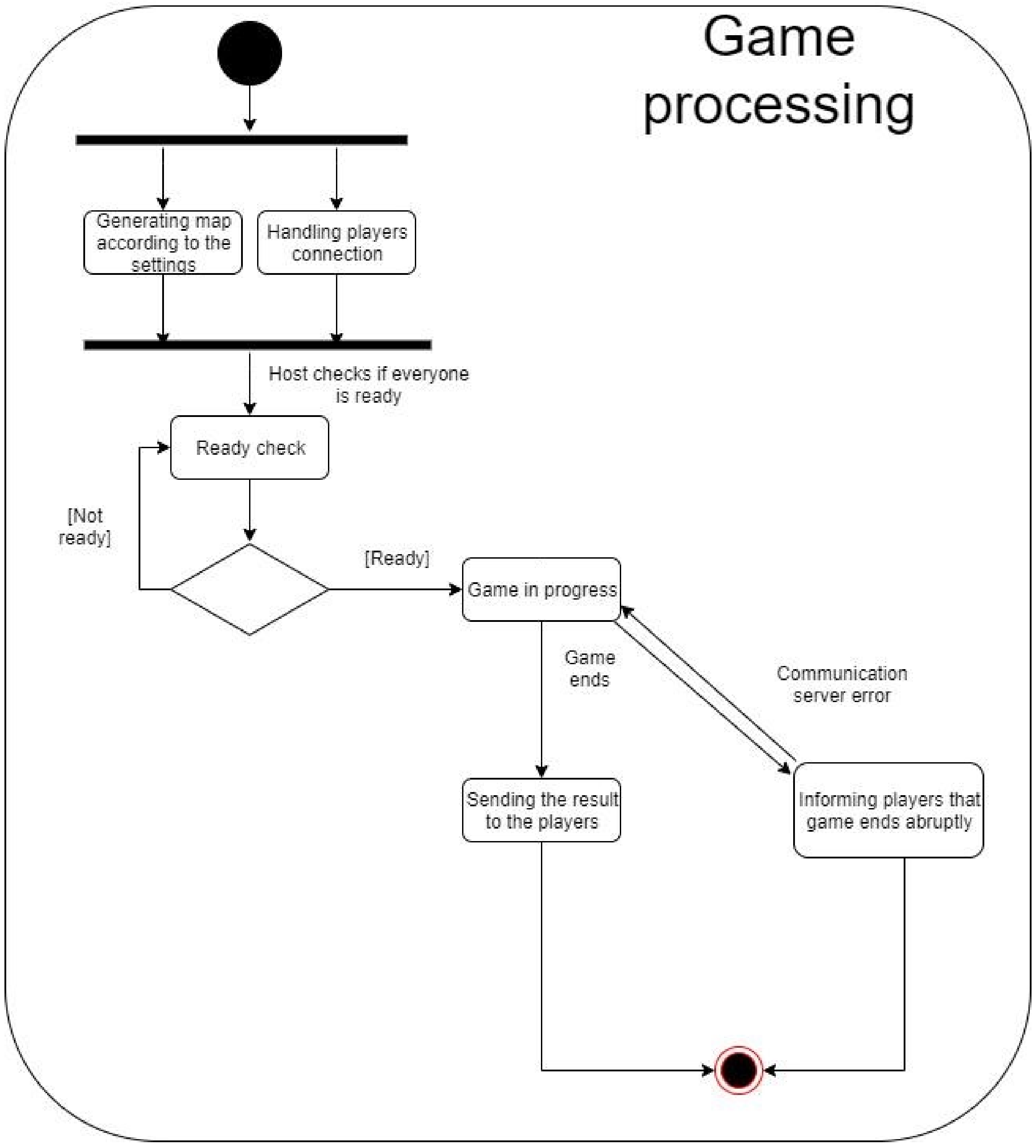
- Self check mode: try block in the code
- Self repair mode: catch block
- Safe mode: person who is responsible for Gm or/and CS checks console/s for errors if it can be fixed fix this. Safe mode can be skipped if error is marked as critical.
- Backup mode: If possible GM/CS sends error message to all players and CS/GM, also saves logs with error message as title.

Additional comment

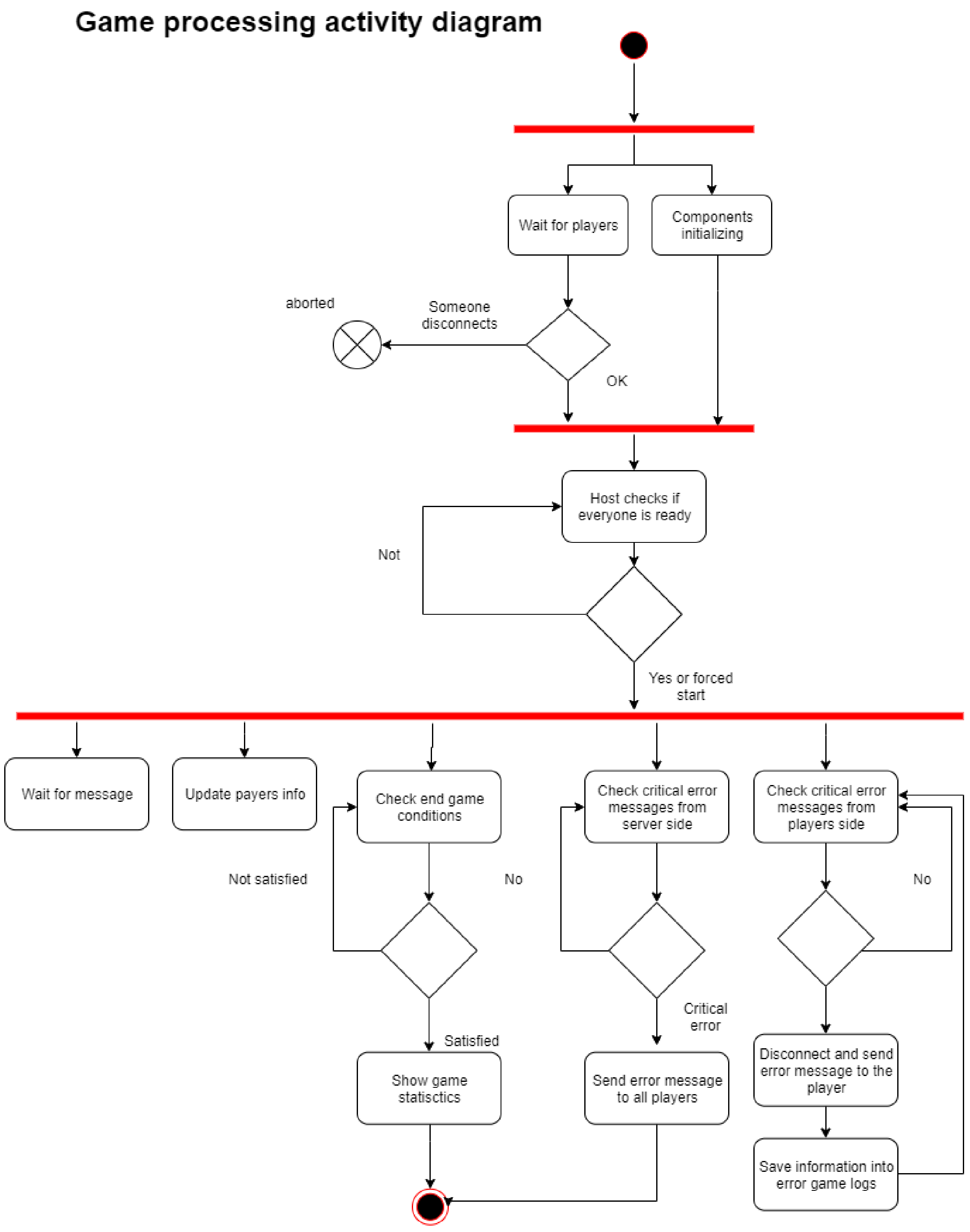
- If game master fails to update player’s info 5 times consecutively then game master remove him from the game (game master stops to send and get messages from him).
- Reconnect will lead to response message: "sorry you were removed by game master"

Special system states description of server side

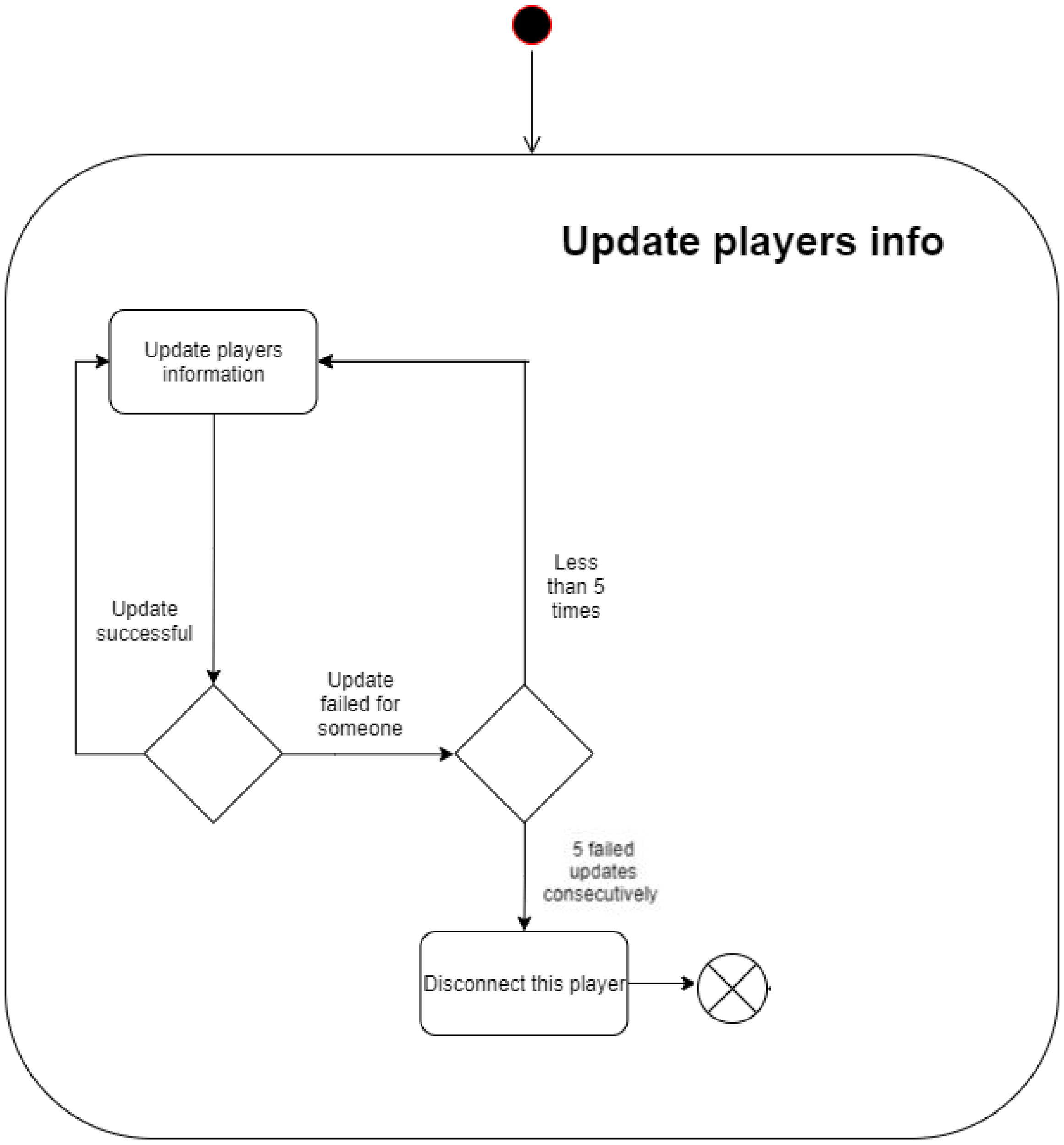



Game processing
Generating map according to the settings
Handling players connection
Host checks if everyone is ready
Ready check
[Not ready]
[Ready]
Game in progress
Game ends
Communication server error
Sending the result to the players
Informing players that game ends abruptly

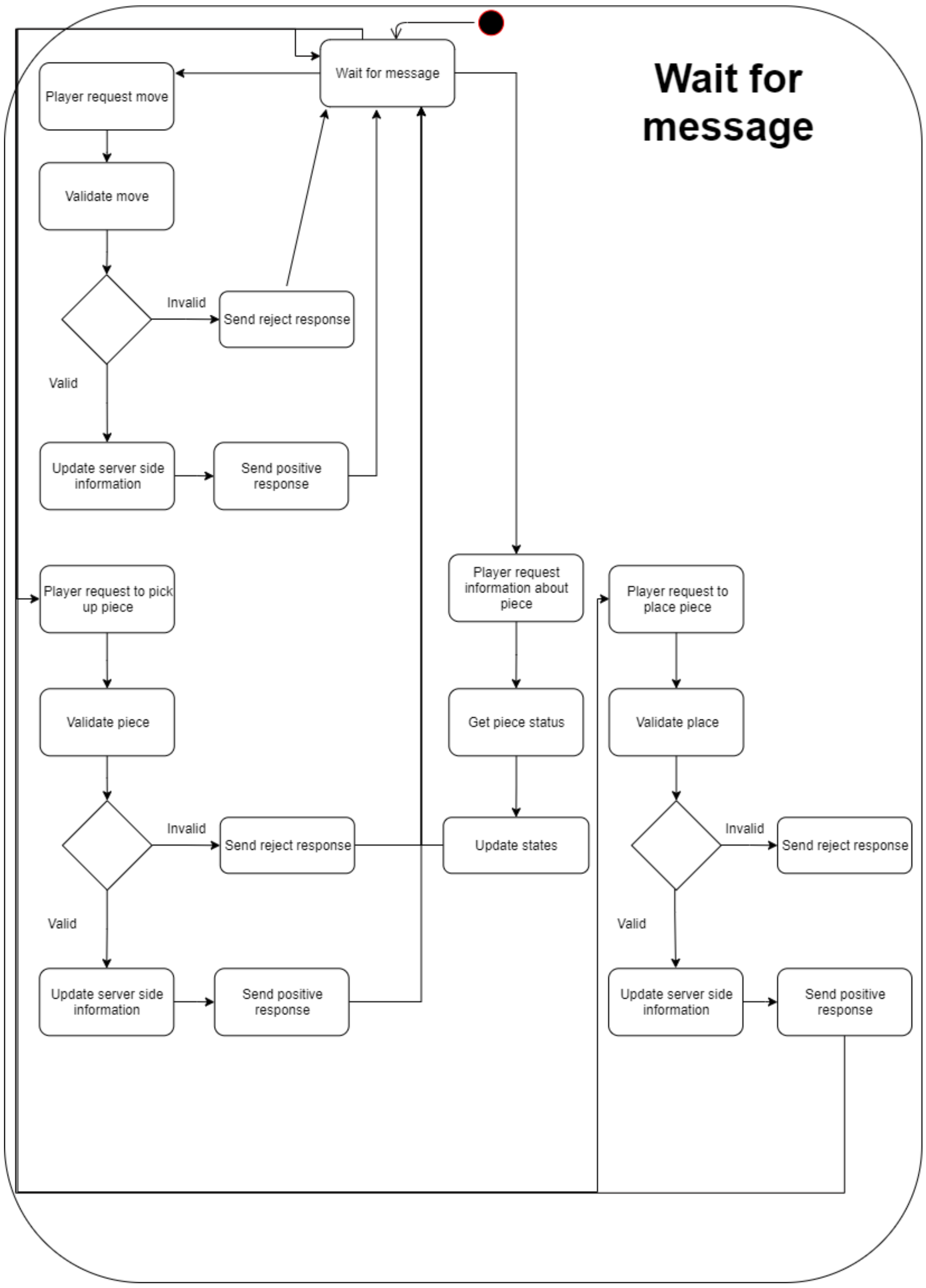
Game process
We had one activity diagram and we decided to divide it into three parts , the second diagram and the third was introduced as a state in game process diagram


Game processing activity diagram
Wait for players
Components initializing
aborted
Someone disconnects
OK
Host checks if everyone is ready
Not
Yes or forced start
Wait for message
Update payers info
Check end game conditions
Check critical error messages from server side
Check critical error messages from players side
Not satisfied
No
No
Satisfied
Critical error
Show game statisctics
Send error message to all players
Disconnect and send error message to the player
Save information into error game logs

Game play


Update players info
Update players information
Update successful
Update failed for someone
Less than 5 times
5 failed updates consecutively
Disconnect this player

Game termination



4.4 Game related objects

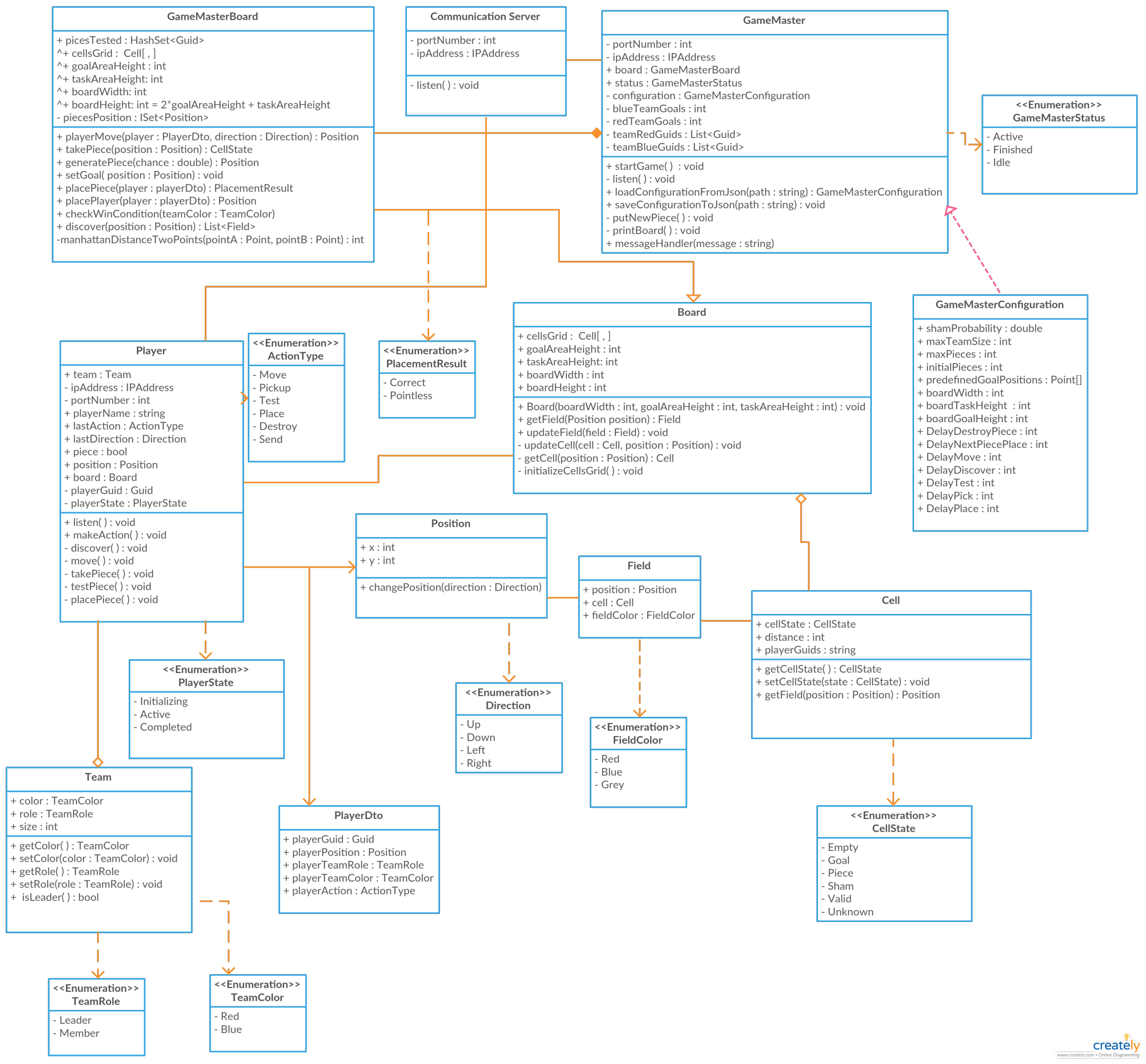
- **Board** - Object which is generated in the beginning of the game and handled game along by Game Master. It provides the players with the information of the entire current state of the game and whenever some changes occur Game Master updates the board. The minimum dimension of the board is 4x4 and maximum dimension is 32x32.
- **Player** - The participant of the game. As a game object is used for collecting the information for each player, its current state and all achievements.
- **Host** - The player which handles the launch of the game when all the players are ready. Its role is to create a lobby and start the game.
- **Team** - The collection of players no more than 4 and only 2 teams playing at the same time. Each team has its own team leader which is chosen by all the players. The color of the team can be chosen by team leader to be Red or Blue.
- **Piece** - The object which is used as award in the game appearing randomly in the board and supposed to be discovered by the players as soon as possible. Piece has two types: 'good' and 'sham'. This object can have different states depending on whether its picked by the player or not and etc.
- **Goal** - The part of the Board, which is different for each of the teams. The player can interact with it by bringing the found pieces. The information about the current state of the Goal is saved in this object.

4.5 Additional information

- player Name: it is configured when player opens his/her console he/she must enter his/her name.
- Max number of pieces: max number of pieces that can be simultaneously at red/blue game area .
- Valid cellstate: tested piece.
- Unknown cellstate: means that the player don't know the state of this field
- initialPieces: If after piece was removed situation would be curNumOfPieces smaller than maxNumOfPieces then new piece is placed(for example: initialPieces is 3 and MaxNumOfPieces is 2 and 30 seconds passed

and players did not find pieces then remove 3 pieces and place 2 new ones).

- Also when piece is placed new one will appear but if and only if curNumOfPieces smaller than maxNumOfPieces.
- Players are assigned to a team in the following manner: we have RP = 0 and BP = 0 then when player connects add to red team then when second player connects add to blue team and so on till limit.
- **In CellState: Sham is removed.**
- **In Gamemaster board : method generatePiece has been changed.**
- **Now when player puts or tests piece there is chance that piece is sham. Previously it was implemented during piece generation.**
- **PiecesTested : Hashset<Guid> is added to GMboard.**
- **When you test piece and its not sham GM player is added to HashSet (or any other structure) Now if player wants to test it one more time we deny this request and when player puts the piece we can check if it was tested or not.**


GameMasterBoard
+ picesTested : HashSet<Guid>
^+ cellsGrid : Cell[,]
^+ goalAreaHeight : int
^+ taskAreaHeight: int
^+ boardWidth: int
^+ boardHeight: int = 2*goalAreaHeight + taskAreaHeight
- piecesPosition : ISet<Position>
+ playerMove(player : PlayerDto, direction : Direction) : Position
+ takePiece(position : Position) : CellState
+ generatePiece(chance : double) : Position
+ setGoal(position : Position) : void
+ placePiece(player : playerDto) : PlacementResult
+ placePlayer(player : playerDto) : Position
+ checkWinCondition(teamColor : TeamColor)
+ discover(position : Position) : List<Field>
-manhattanDistanceTwoPoints(pointA : Point, pointB : Point) : int
Communication Server
- portNumber : int
- ipAddress : IPAddress
- listen( ) : void
GameMaster
- portNumber : int
- ipAddress : IPAddress
+ board : GameMasterBoard
+ status : GameMasterStatus
- configuration : GameMasterConfiguration
- blueTeamGoals : int
- redTeamGoals : int
- teamRedGuids : List<Guid>
- teamBlueGuids : List<Guid>
+ startGame( ) : void
- listen( ) : void
+ loadConfigurationFromJson(path : string) : GameMasterConfiguration
+ saveConfigurationToJson(path : string) : void
- putNewPiece( ) : void
- printBoard( ) : void
+ messageHandler(message : string)
<<Enumeration>> GameMasterStatus
- Active
- Finished
- Idle
Player
+ team : Team
- ipAddress : IPAddress
- portNumber : int
+ playerName : string
+ lastAction : ActionType
+ lastDirection : Direction
+ piece : bool
+ position : Position
+ board : Board
- playerGuid : Guid
- playerState : PlayerState
+ listen( ) : void
+ makeAction( ) : void
- discover( ) : void
- move( ) : void
- takePiece( ) : void
- testPiece( ) : void
- placePiece( ) : void
<<Enumeration>> ActionType
- Move
- Pickup
- Test
- Place
- Destroy
- Send
<<Enumeration>> PlacementResult
- Correct
- Pointless
Board
+ cellsGrid : Cell[,]
+ goalAreaHeight : int
+ taskAreaHeight: int
+ boardWidth : int
+ boardHeight : int
+ Board(boardWidth : int, goalAreaHeight : int, taskAreaHeight : int) : void
+ getField(Position position) : Field
+ updateField(field : Field) : void
- updateCell(cell : Cell, position : Position) : void
- getCell(position : Position) : Cell
- initializeCellsGrid( ) : void
GameMasterConfiguration
+ shamProbability : double
+ maxTeamSize : int
+ maxPieces : int
+ initialPieces : int
+ predefinedGoalPositions : Point[]
+ boardWidth : int
+ boardTaskHeight : int
+ boardGoalHeight : int
+ DelayDestroyPiece : int
+ DelayNextPiecePlace : int
+ DelayMove : int
+ DelayDiscover : int
+ DelayTest : int
+ DelayPick : int
+ DelayPlace : int
Position
+ x : int
+ y : int
+ changePosition(direction : Direction)
Field
+ position : Position
+ cell : Cell
+ fieldColor : FieldColor
Cell
+ cellState : CellState
+ distance : int
+ playerGuids : string
+ getCellState( ) : CellState
+ setCellState(state : CellState) : void
+ getField(position : Position) : Position
<<Enumeration>> PlayerState
- Initializing
- Active
- Completed
<<Enumeration>> Direction
- Up
- Down
- Left
- Right
<<Enumeration>> FieldColor
- Red
- Blue
- Grey
Team
+ color : TeamColor
+ role : TeamRole
+ size : int
+ getColor( ) : TeamColor
+ setColor(color : TeamColor) : void
+ getRole( ) : TeamRole
+ setRole(role : TeamRole) : void
+ isLeader( ) : bool
PlayerDto
+ playerGuid : Guid
+ playerPosition : Position
+ playerTeamRole : TeamRole
+ playerTeamColor : TeamColor
+ playerAction : ActionType
<<Enumeration>> CellState
- Empty
- Goal
- Piece
- Sham
- Valid
- Unknown
<<Enumeration>> TeamRole
- Leader
- Member
<<Enumeration>> TeamColor
- Red
- Blue
creately

4.6 Game rules

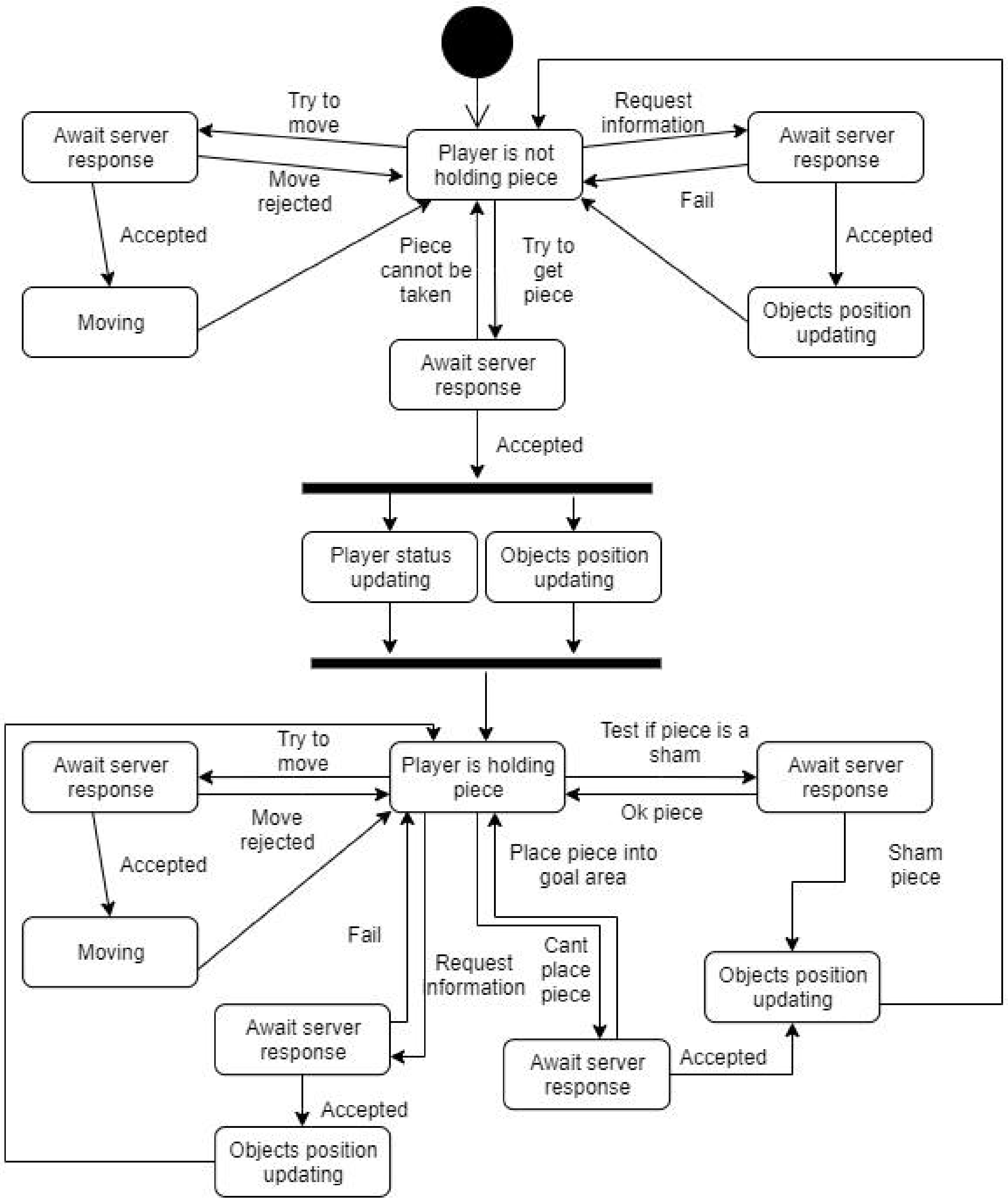
- The Host of the game creates lobby (one lobby only) for the Players and starts the game when all of the team members are ready.
- Players have to have ID of 128-bits. (GUID) Example:https://docs.microsoft.com/en-us/dotnet/api/system.guid
- Each component of the game assigns for itself guid, after object creation.
- After the launch of the game Game Master generates the Board, randomly places Pieces and Players.
- Players are placed in the Goal area of their Team.

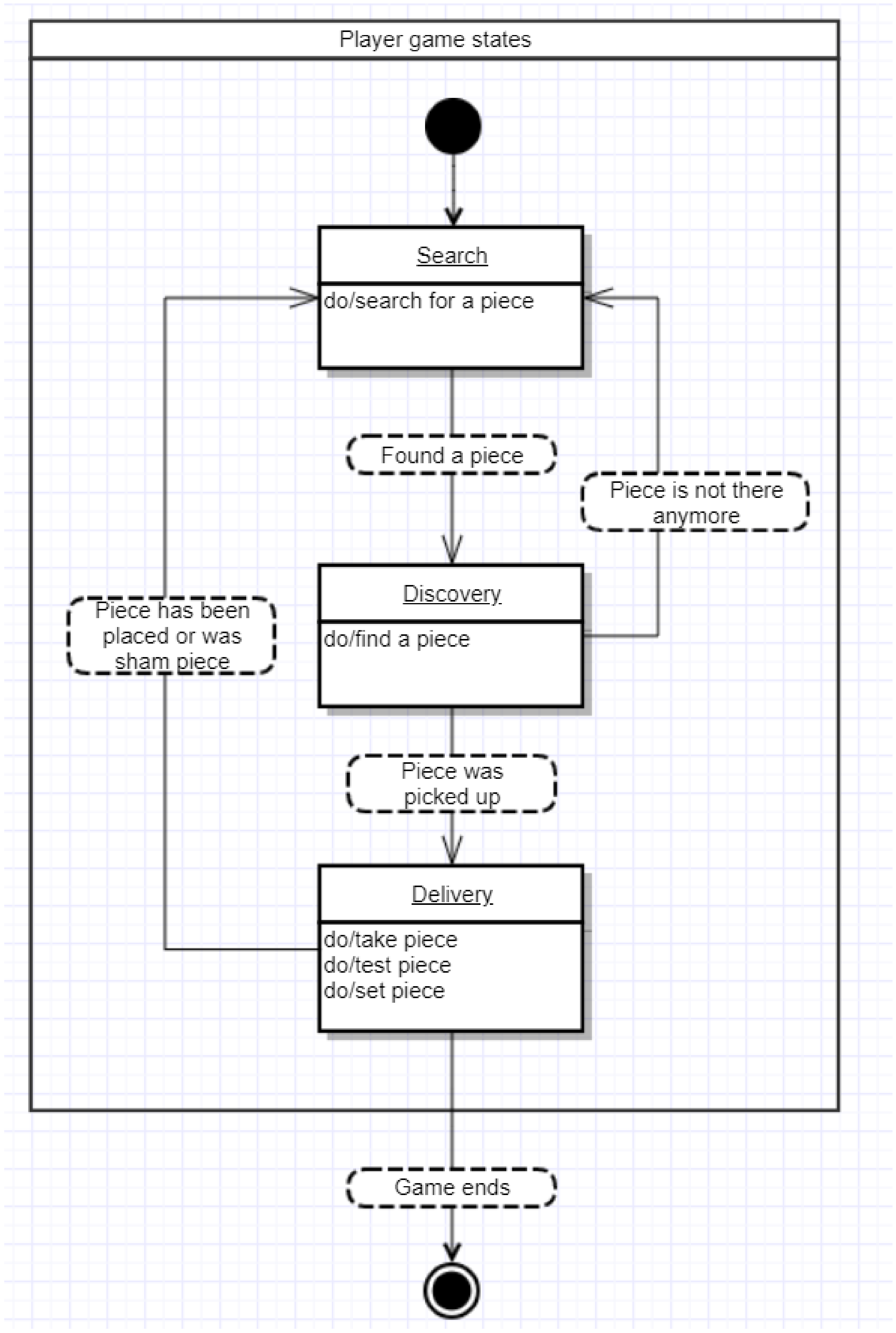
- The initial goal of each Team is to bring some exact amount of Pieces to the Goal area and put them in the right places.
- Each player can carry one piece only at a time.
- Pieces are dropped every 30 seconds and the previous ones disappear after this time expires if they are not approached by player till that moment.
- Each Player can distinguish the content of 8 neighboring fields, other parts of the Board are hidden.
- Player steps out of the goal area and finds out the Manhattan distance to the nearest Piece.
- Player is not allowed to step to the other players field and go to the Goal area of the opposite Team.
- Upon approach a Piece a Player can choose either to bring the piece back without testing it or check if the piece is good or sham.
- If the piece is brought to the Goals area by the Player, he/she needs to place the piece in the right field. If the Player places a piece correctly, he/she will be notified that one of the goals have been completed.
- If the Piece was placed incorrectly, the Player will be notified that they discovered a non-goal.
- If the Piece turns out to be sham, the player gets no information.
- The team that finds the right Pieces and puts in the right fields first wins and the game finishes.
- The game is won by one only one team.
- The game is about completing the goals.
- Only one game per one communication server and game master and player at the same time.

4.6.1 Additional description

  - Player can send only one message/request and then wait until getting response or till expire time.

Description of the board and player's states during game




Player game states
Search
do/search for a piece
Found a piece
Piece is not there anymore
Discovery
do/find a piece
Piece has been placed or was sham piece
Piece was picked up
Delivery
do/take piece
do/test piece
do/set piece
Game ends

4.7 Game process

- The server starts loading and if it is in operate condition Players are able to connect.
- Each player has opportunity to choose, whether to join a game or create own lobby, so this Player becomes a host.
- When players are loaded in a lobby, then host can start a game if all Players are ready and there are more than 1 Player in each team. Number of Players in each team has to be equal.
- When host is ready to start the game, Game Master generates the Board, randomly places Pieces and Players and all the players are getting loaded on a play board. Then host can start the timer of 30 seconds before the players can move or this timer starts automatically after all the players are loaded.
- Game starts. In order to win a Team need to bring some exact number of true pieces in the goal area faster then their opponents. The real state of the piece is known only to the Game Master and in order to check it player should bring that piece in the goal area. Then team gets a point if piece appears to be true.
- Players need to put true pieces in the correct places and fill all the gaps of the goal area.
- Pieces are dropped every 30 seconds and they disappear if they were not approached by any Player during some stated period of time.
- Player can see only 8 blocks around, other parts of the Board are hidden from the Player and need to be discovered in order to realise if there is a piece over there. However, player knows the distance to the nearest piece which is written in the blocks surrounding each player. It can only be seen by Player.
- There is no chance for player to step on the field of other Player as to step in the opposite team's Goal Area. It is forbidden.

- Whenever Player approaches Piece there can be two decisions to be made. The first is to take Piece without checking it and bring to the Goal Area. The other is to check the piece where it was found to know if it is sham or not. In the first place Player is risking to bring a sham Piece and spend time meaninglessly, but at the same time if this Piece, actually, appears to be true Player saves time.

- If Player decides to check the Piece and it appears to be sham, then it can be left where it stays. Player can inform other Team members that Piece that was found is sham, so others wouldn't spend time on that Piece.

- When Player brings True piece to the Goal area it is crucial to put in the correct place. If everything done correctly all the Team members get notified that one of the Goals was completed. If the Piece was place incorrectly, the Player will be notified that the placement haven't advanced them closer to reaching the Goal. If the Piece turns out to be sham, the player gets no information.

- Players play till one of the following cases take place. The first one is one team completes the Goal Area and they win a game and it gets successfully finished.

- The second option is that one of the teams surrenders and the other team wins. And the last option is that all the Players disconnect from the server, in this case lobby closes, game process terminates.

- 1-4 players in each time, we can have situation even 1 vs 4 or3 vs 2. If we have situation:1 in blue team, 2 in red team and game in progress if player from blue team disconnects (0 players) then red team wins the game.

4.8 Game Interactions

Interaction between Players and Game Master are meant to be on request-response basis. So if a Player needs to interact with the Game Master, then it sends the request and Game Master makes up some response on this request.

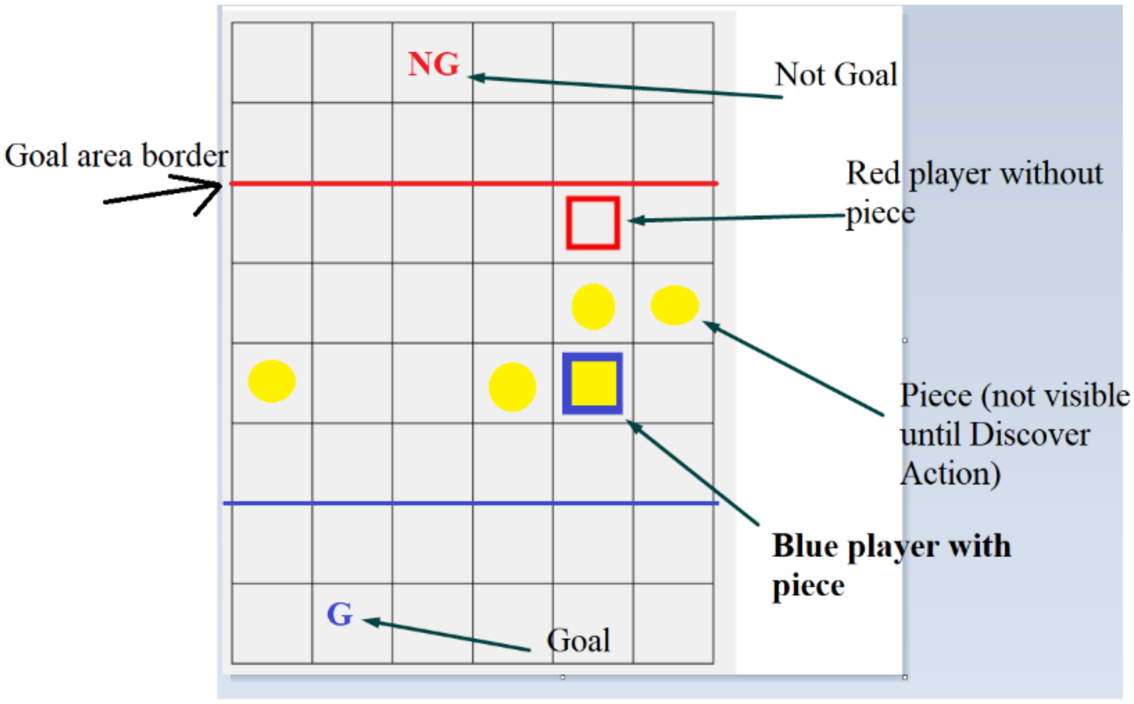
5 Special Features

5.1 GUI

- Still no GUI for CS/GM.
- For Players if you want you can implement it and we don't care how it will be done. For example, we will use either wpf or winforms. However, not everyone chose c sharp for development that's why we won't indicate it in project documentation.
- We can have two options:

1. Stay on "No GUI side" and complete your project using console ASCII power.

2. Interpret console using GUI Example.



5.2 Hardware system exception

- Linux, macOS, dendy, etc is not forbidden but since main system is windows, you must be sure that your solution will be compatible with the solutions of other teams from your group because of 3rd deadline (Cooperation 20 percent).