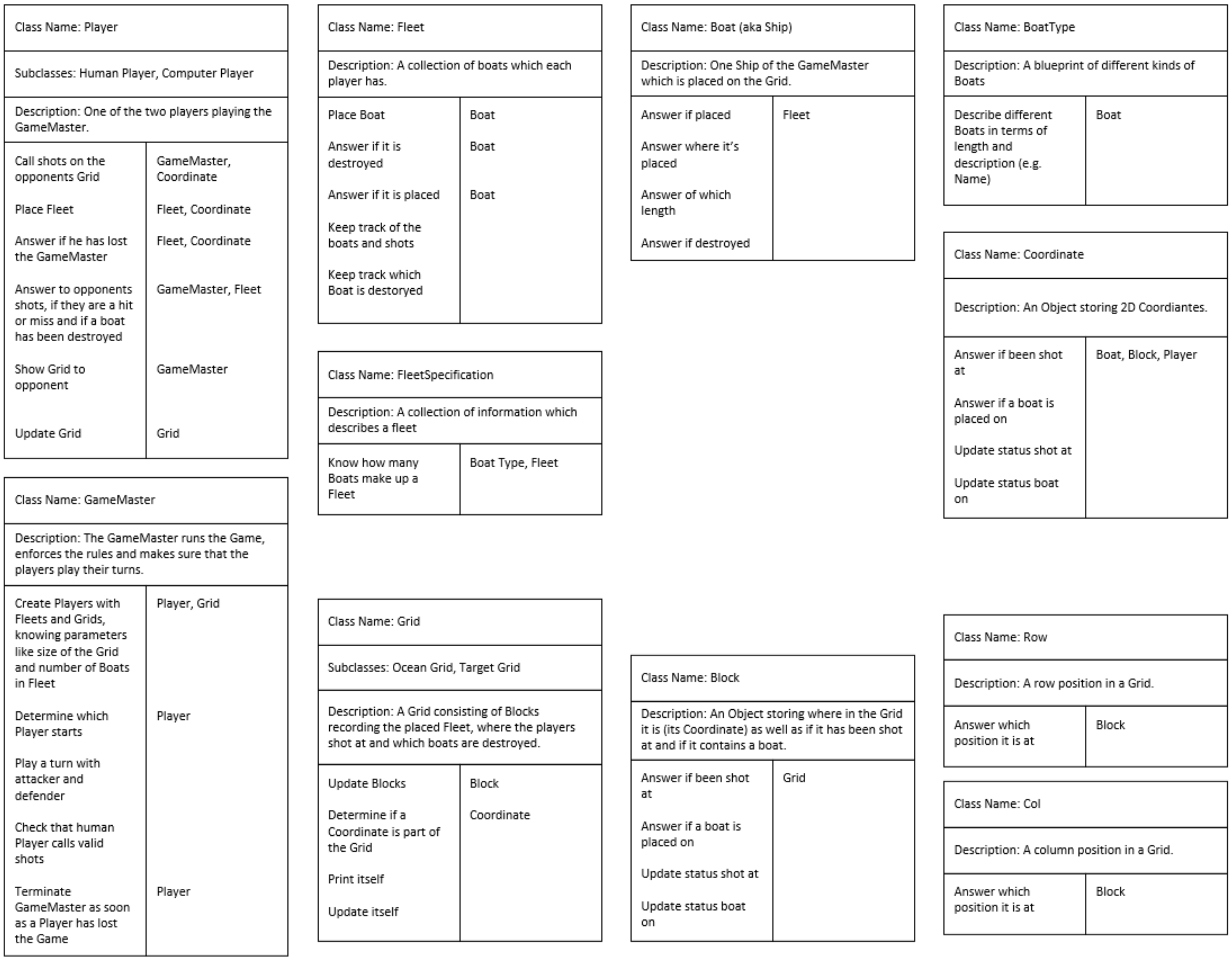
Exercise_01

Ramona Walker, Dominik Johann Arnold, Mark Woolley, Otto Buck

November 2022

1 CRC Cards

These are the CRC Cards as we created them when we started.



Class Name: Player	
Subclasses: Human Player, Computer Player	
Description: One of the two players playing the GameMaster.	
Call shots on the opponents Grid	GameMaster, Coordinate
Place Fleet	Fleet, Coordinate
Answer if he has lost the GameMaster	Fleet, Coordinate
Answer to opponents shots, if they are a hit or miss and if a boat has been destroyed	GameMaster, Fleet
Show Grid to opponent	GameMaster
Update Grid	Grid

Class Name: GameMaster	
Description: The GameMaster runs the Game, enforces the rules and makes sure that the players play their turns.	
Create Players with Fleets and Grids, knowing parameters like size of the Grid and number of Boats in Fleet	Player, Grid
Determine which Player starts	Player
Play a turn with attacker and defender	
Check that human Player calls valid shots	
Terminate GameMaster as soon as a Player has lost the Game	Player

Class Name: Fleet	
Description: A collection of boats which each player has.	
Place Boat	Boat
Answer if it is destroyed	Boat
Answer if it is placed	Boat
Keep track of the boats and shots	
Keep track which Boat is destoryed	

Class Name: FleetSpecification	
Description: A collection of information which describes a fleet	
Know how many Boats make up a Fleet	Boat Type, Fleet

Class Name: Grid	
Subclasses: Ocean Grid, Target Grid	
Description: A Grid consisting of Blocks recording the placed Fleet, where the players shot at and which boats are destroyed.	
Update Blocks	Block
Determine if a Coordinate is part of the Grid	Coordinate
Print itself	
Update itself	

Class Name: Boat (aka Ship)	
Description: One Ship of the GameMaster which is placed on the Grid.	
Answer if placed	Fleet
Answer where it's placed	
Answer of which length	
Answer if destroyed	

Class Name: Block	
Description: An Object storing where in the Grid it is (its Coordinate) as well as if it has been shot at and if it contains a boat.	
Answer if been shot at	Grid
Answer if a boat is placed on	
Update status shot at	
Update status boat on	

Class Name: BoatType	
Description: A blueprint of different kinds of Boats	
Describe different Boats in terms of length and description (e.g. Name)	Boat

Class Name: Coordinate	
Description: An Object storing 2D Coordiantes.	
Answer if been shot at	Boat, Block, Player
Answer if a boat is placed on	
Update status shot at	
Update status boat on	

Class Name: Row	
Description: A row position in a Grid.	
Answer which position it is at	Block

Class Name: Col	
Description: A column position in a Grid.	
Answer which position it is at	Block

cards.png

2 Following the Responsibility Driven Design, describe the main classes you designed to be your project in terms of responsibilities and collaborations.

- **Game / GameMaster**
  GameMaster makes sure the game is played according to the rules and has two players. The class is responsible for stopping the game when it's over.
  Responsibilities: Initialize the game and ensure the correct execution of the game rules. Determine the starting player and make sure the game ends when a player has lost.
  Collaborations: Mainly interact with the *Players* of which the game consists of. Interact directly with the Players *Grid* when the GameMaster must print the state of the game. GameMaster uses *Coordinates* to play a turn.

- **Player (Subclass PlayerHuman, PlayerComputer)**
  A Player has ONE Grid and ONE Fleet. He collaborates with the GameMaster by declaring and receiving shots. He collaborates with his Fleet and Grid during placement of Boats and while declaring and receiving shots.
  Note: Difference between Human and Computer is only how they place their Fleet and how they call shots. Computer is randomized (or any strategy really), Human is via user input.
  Responsibilities: Must place his Fleet, which leads to Boats being placed. Must be able to call valid shots for the other player as well as receive them from the other player (via GameMaster). Simultaneously, whenever the Fleet is changed or shots are recorded, the player must update his Fleet and Grid accordingly. Furthermore, the player must be able to provide his up-to-date Grid to the GameMaster and he must be able to declare whether he has lost the game.
  Collaborations: With the *GameMaster* when supplying the Grid. With the *Fleet* in all actions regarding placement and shots receiving. When placing the fleet, *Boat* in Fleet is called. With the *Grid* during all the actions which lead to the player updating his Grid. A list of *Coordinates* is used to store received and taken shots.

- **Fleet**
  A Fleet is a collection of Boats and keeps track of placed and destroyed Boats. The specification of the collection is set in the game rules.
  Responsibilities: At any point declare if the whole Fleet is destroyed. The Fleet can receive a shot and declare if any Boat is hit. Must be able to place a Boat when given a list of Coordinates and must be able to tell for a given list of Coordinates if there is an overlap in boats (e.g. if there is a boat already).
  Collaborations: With the *Boat* and *Coordinate* when a Boat is placed and when the status (destroyed) is determined. With *FleetSpecification* to receive the specific boats it consists of.

- **Grid**
  A Grid is a collection of Blocks, given the size of the game rules. It must be able to print itself as either target or ocean grid and needs to update its Blocks according to different events.
  Responsibilities: The Grid must provide a printing method as either a target or ocean grid. It must be able to record and remember shots taken at it and boats placed on it.
  Collaborations: With the *Block* whenever a block needs to get updated. The Player will ask the Grid to update for different events. The Grid will pass on this Information to the correct Block. Therefore it also needs to collaborate with *Coordinate.* To print it needs to collaborate with *GameUtils.* (Changing integers to coordinates and getting game size.)

- **Block**
  A Block is the elementary unit of the grid. Each block has a position in the grid in form of a Coordinate. Furthermore, it knows its state regarding if it has been shot at, it has a boat placed on itself, if it should act as destroyed and what the printing character is.
  Note: Not all states are important for both printing options.
  Responsibilities: Know and update the status regarding shot at, boat placed on, destroyed status and what kind of character to print if a boat is on there. Need to be able to be called to change any state.
  Collaborations: With the *Coordinate* to store its location.

- **Coordinate**
  The Coordinate is the fundamental building block of the whole game and is used through all classes. It provides a common language for positions which make up Blocks, shots, and anything else in the game. It can also convert between internal integer representation of rows and columns and the external representation (e.g. A2)
  Responsibilities: Be able to tell the position in a 2D plane with row and columns numbers. Must be able to print

itself either in integer representation or in external representation
Collaborations: With *GameUtils* to convert integer-representation to letter-representation.

- **FleetSpecification**
This class is a lookup artifact which specifies what number of different BoatTypes a Fleet consists of.
Responsibilities: Know the exact number of BoatTypes which make up a Fleet. Give that Information to Fleet.
Collaborations: With *Boattype* to define what boats are in a fleet.

- **BoatType**
This class describes the different types of Boats that exist in terms of name, description, and length.
Responsibilities: Know what kind of Boats exist and provide this information.
Collaborations: none

3 Why do you consider the other classes as less important? Following the Responsibility Driven Design, reflect if some of those non-main classes have similar/little responsibility and could be changed, merged, or removed.

- **OceanGrid, TargetGrid**
  We realized that the distinction between two different grids is superfluous. It's enough that each Player only has one Grid and the Grid itself must be able to present itself as either target or ocean grid.
- **Row / Column Types**
  We decided to not use enum types to depict rows and columns and use simple integers instead. Reasoning: The most natural way to describe rows and columns as coordinates are in fact integers, not enum types.
- **FleetSpecification**
  The FleetSpecificaiton could also be set in the game rules, but we decided to make it a dedicated enum type. We think that it makes the organization of the code cleaner.

4 Draw the class diagram of the aforementioned main elements of your game.

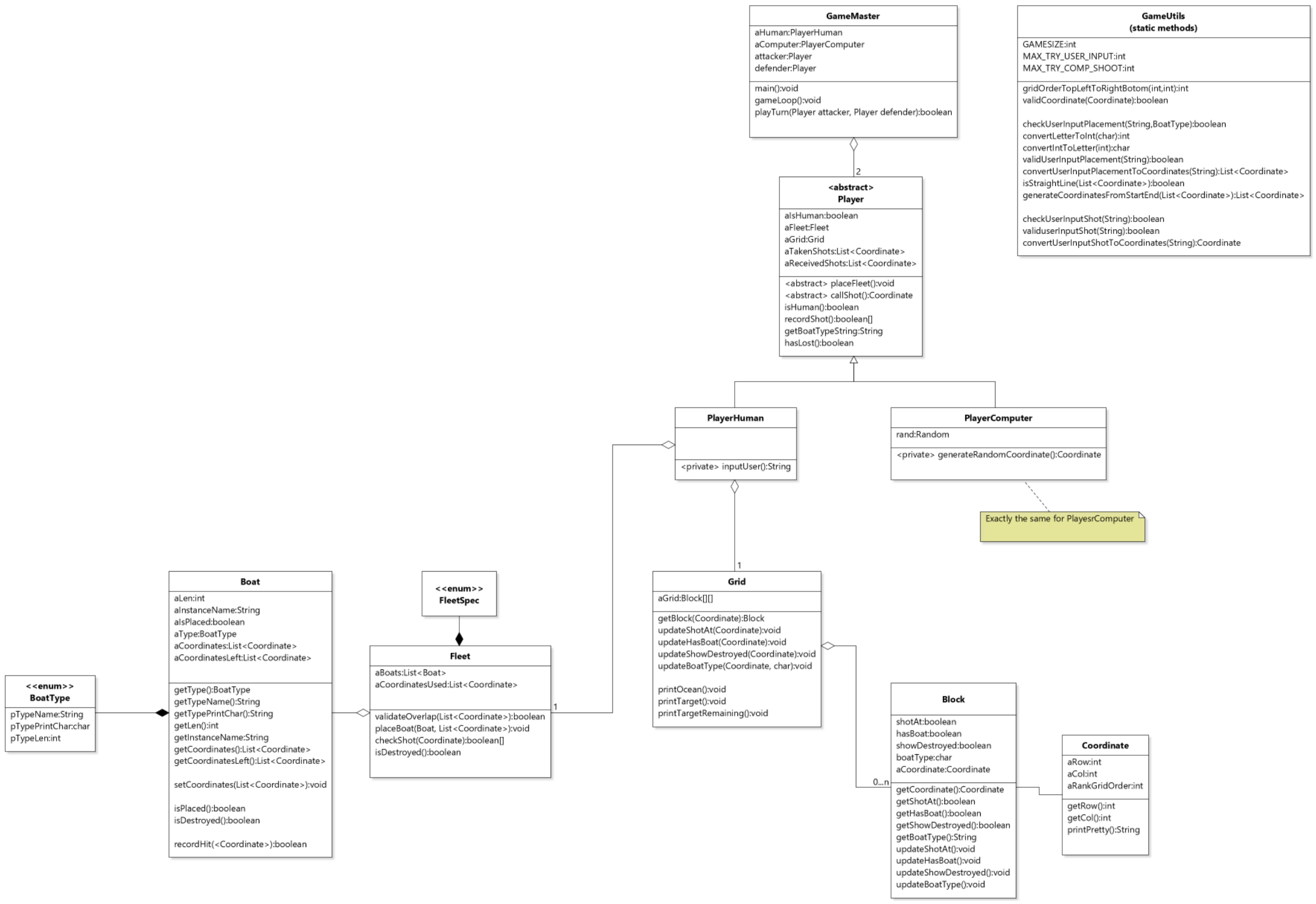

GameMaster
aHuman:PlayerHuman
aComputer:PlayerComputer
attacker:Player
defender:Player
main():void
gameLoop():void
playTurn(Player attacker, Player defender):boolean
GameUtils
(static methods)
GAMESIZE:int
MAX_TRY_USER_INPUT:int
MAX_TRY_COMP_SHOOT:int
gridOrderTopLeftToRightBotom(int,int):int
validCoordinate(Coordinate):boolean
checkUserInputPlacement(String,BoatType):boolean
convertLetterToInt(char):int
convertIntToLetter(int):char
validUserInputPlacement(String):boolean
convertUserInputPlacementToCoordinates(String):List<Coordinate>
isStraightLine(List<Coordinate>):boolean
generateCoordinatesFromStartEnd(List<Coordinate>):List<Coordinate>
checkUserInputShot(String):boolean
validuserInputShot(String):boolean
convertUserInputShotToCoordinates(String):Coordinate
2
<abstract>
Player
aIsHuman:boolean
aFleet:Fleet
aGrid:Grid
aTakenShots:List<Coordinate>
aReceivedShots:List<Coordinate>
<abstract> placeFleet():void
<abstract> callShot():Coordinate
isHuman():boolean
recordShot():boolean[]
getBoatTypeString:String
hasLost():boolean
PlayerHuman
<private> inputUser():String
PlayerComputer
rand:Random
<private> generateRandomCoordinate():Coordinate
Exactly the same for PlayesrComputer
1
Grid
aGrid:Block[][]
getBlock(Coordinate):Block
updateShotAt(Coordinate):void
updateHasBoat(Coordinate):void
updateShowDestroyed(Coordinate):void
updateBoatType(Coordinate, char):void
printOcean():void
printTarget():void
printTargetRemaining():void
Block
shotAt:boolean
hasBoat:boolean
showDestroyed:boolean
boatType:char
aCoordinate:Coordinate
0...n
getCoordinate():Coordinate
getShotAt():boolean
getHasBoat():boolean
getShowDestroyed():boolean
getBoatType():String
updateShotAt():void
updateHasBoat():void
updateShowDestroyed():void
updateBoatType():void
Coordinate
aRow:int
aCol:int
aRankGridOrder:int
getRow():int
getCol():int
printPretty():String
<<enum>>
BoatType
pTypeName:String
pTypePrintChar:char
pTypeLen:int
Boat
aLen:int
aInstanceName:String
aIsPlaced:boolean
aType:BoatType
aCoordinates:List<Coordinate>
aCoordinatesLeft:List<Coordinate>
getType():BoatType
getTypeName():String
getTypePrintChar():String
getLen():int
getInstanceName:String
getCoordinates():List<Coordinate>
getCoordinatesLeft():List<Coordinate>
setCoordinates(List<Coordinate>):void
isPlaced():boolean
isDestroyed():boolean
recordHit(<Coordinate>):boolean
<<enum>>
FleetSpec
Fleet
aBoats:List<Boat>
aCoordinatesUsed:List<Coordinate>
validateOverlap(List<Coordinate>):boolean
placeBoat(Boat, List<Coordinate>):void
checkShot(Coordinate):boolean[]
isDestroyed():boolean
1

Figure 1: Class diagram

5 Draw an object diagram to show the main elements of your game in a step of the game of your choosing.

This section includes two object diagrams. The first is a complete state of the game, including all Coordinate objects. The second one depicts the exact same situation, but for clarity leaves out all Coordinate objects. This is meant to help understanding the major elements without cluttering the diagram.

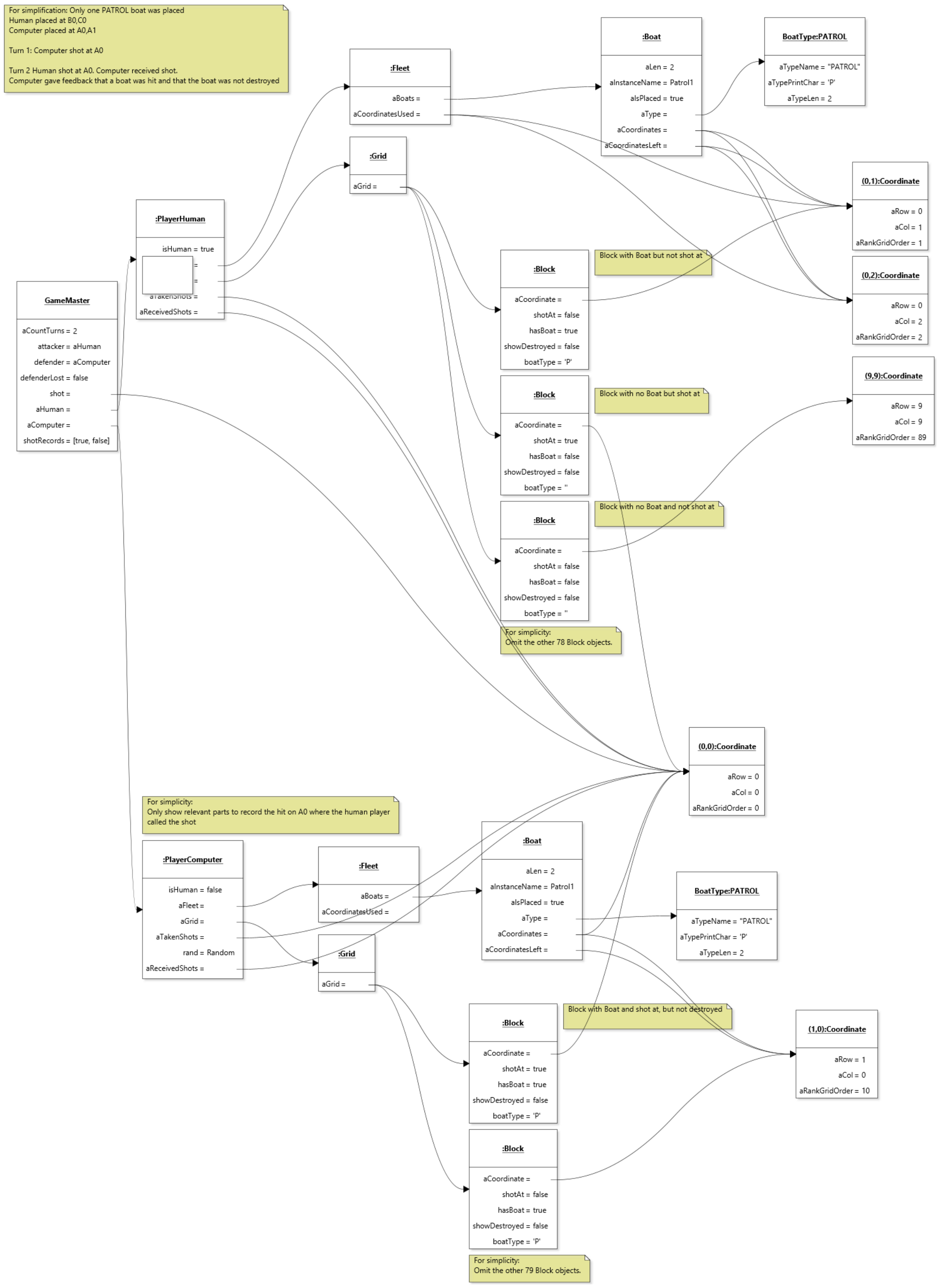


Figure 2: Object diagram including Coordinate objects

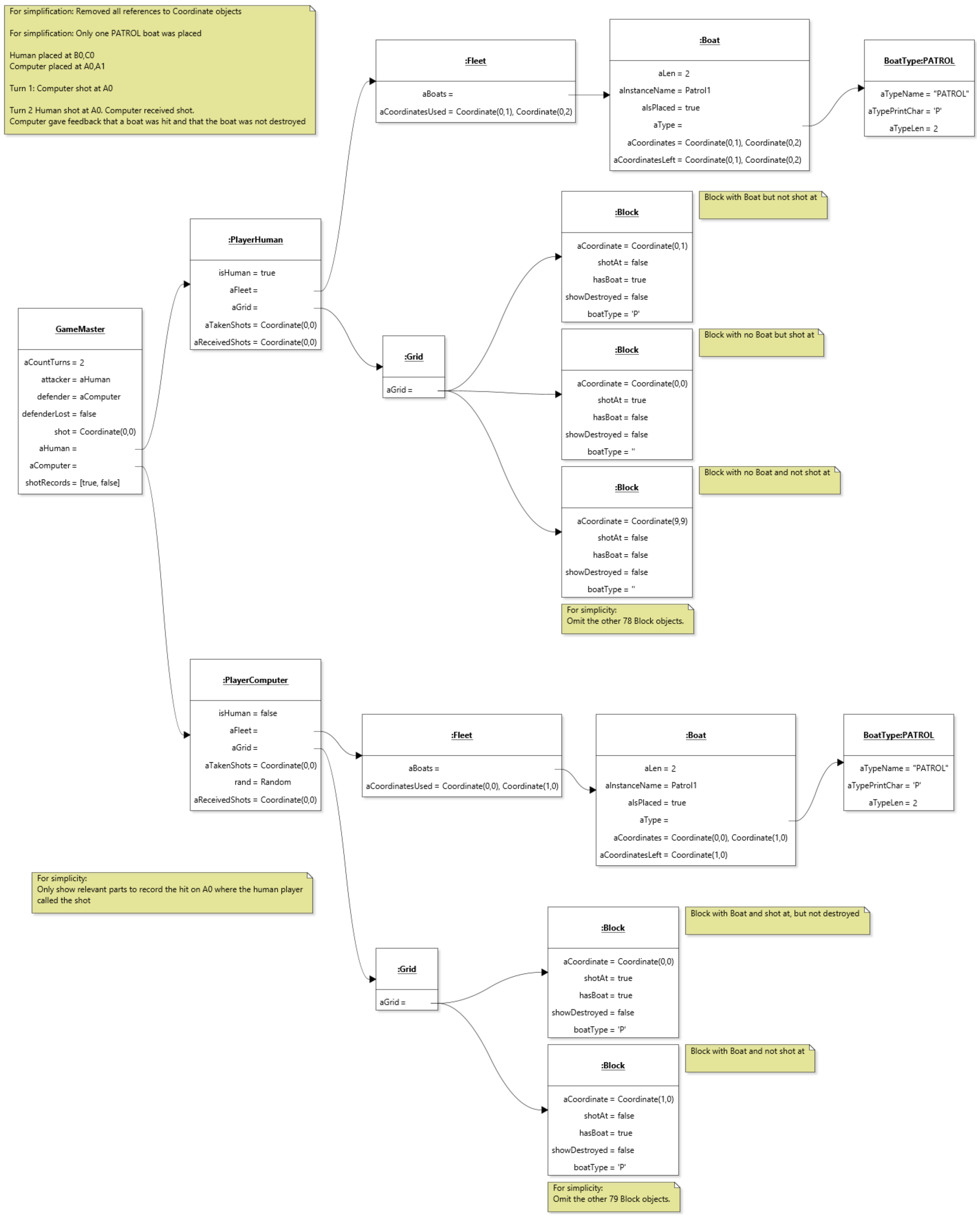

For simplification: Removed all references to Coordinate objects
For simplification: Only one PATROL boat was placed
Human placed at B0,C0
Computer placed at A0,A1
Turn 1: Computer shot at A0
Turn 2 Human shot at A0. Computer received shot.
Computer gave feedback that a boat was hit and that the boat was not destroyed
GameMaster
aCountTurns = 2
attacker = aHuman
defender = aComputer
defenderLost = false
shot = Coordinate(0,0)
aHuman =
aComputer =
shotRecords = [true, false]
:PlayerHuman
isHuman = true
aFleet =
aGrid =
aTakenShots = Coordinate(0,0)
aReceivedShots = Coordinate(0,0)
:Fleet
aBoats =
aCoordinatesUsed = Coordinate(0,1), Coordinate(0,2)
:Boat
aLen = 2
aInstanceName = Patrol1
aIsPlaced = true
aType =
aCoordinates = Coordinate(0,1), Coordinate(0,2)
aCoordinatesLeft = Coordinate(0,1), Coordinate(0,2)
BoatType:PATROL
aTypeName = "PATROL"
aTypePrintChar = 'P'
aTypeLen = 2
:Grid
aGrid =
:Block
aCoordinate = Coordinate(0,1)
shotAt = false
hasBoat = true
showDestroyed = false
boatType = 'P'
Block with Boat but not shot at
:Block
aCoordinate = Coordinate(0,0)
shotAt = true
hasBoat = false
showDestroyed = false
boatType = ''
Block with no Boat but shot at
:Block
aCoordinate = Coordinate(9,9)
shotAt = false
hasBoat = false
showDestroyed = false
boatType = ''
Block with no Boat and not shot at
For simplicity:
Omit the other 78 Block objects.
:PlayerComputer
isHuman = false
aFleet =
aGrid =
aTakenShots = Coordinate(0,0)
rand = Random
aReceivedShots = Coordinate(0,0)
:Fleet
aBoats =
aCoordinatesUsed = Coordinate(0,0), Coordinate(1,0)
:Boat
aLen = 2
aInstanceName = Patrol1
aIsPlaced = true
aType =
aCoordinates = Coordinate(0,0), Coordinate(1,0)
aCoordinatesLeft = Coordinate(1,0)
BoatType:PATROL
aTypeName = "PATROL"
aTypePrintChar = 'P'
aTypeLen = 2
For simplicity:
Only show relevant parts to record the hit on A0 where the human player called the shot
:Grid
aGrid =
:Block
aCoordinate = Coordinate(0,0)
shotAt = true
hasBoat = true
showDestroyed = false
boatType = 'P'
Block with Boat and shot at, but not destroyed
:Block
aCoordinate = Coordinate(1,0)
shotAt = false
hasBoat = true
showDestroyed = false
boatType = 'P'
Block with Boat and not shot at
For simplicity:
Omit the other 79 Block objects.

Figure 3: Object diagram without the Coordinate objects