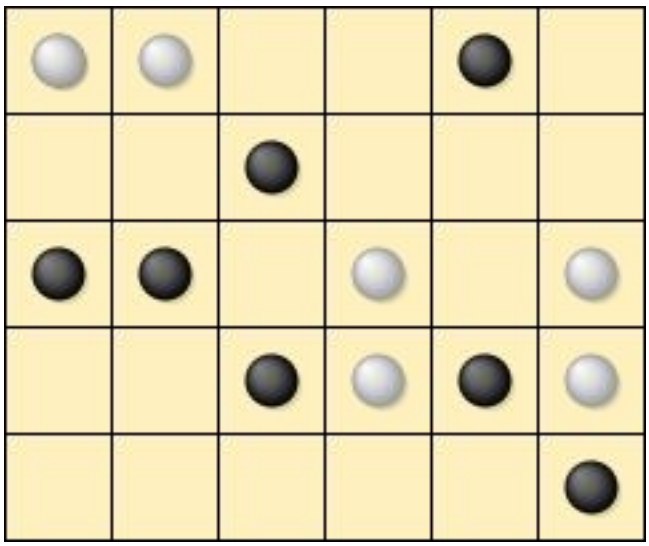
Yoté



Group Members:
Anila Hasaj 0802406
Christian Rei 0832859
Jackson Keenan 0801837
Robert Little 0802985

Table Of Contents

Title Page **1**

Executive Summary **3**

Game Rules **4**

User Cases **5**

CRC Cards **9**

Interface/Type model **11**

Class Diagram **12**

Class Information **13**

Executive Summary

Yoté begins on a empty 5×6 board, and has two players with twelve pieces each. The white player begins, alternating turns choosing to place, move or attack a piece.

High Level Overview:

Yoté is made up of a small number of entities. The key entities are the Board, Player, Cells, Pieces, Rules, Driver,. These are the nouns needed to run the game efficiently and have the game be interchangeable with other board games. The entities communicate with each other in different ways like one or two way communication to give and take information to follow to rules.

Limitations:

A player can only place one piece at a time after their opponents turn is complete. A player cannot place a piece on an occupied cell. A player cannot place with no pieces left in hand. Attacks are only carried out by jumping over a piece. An attack cannot happen if the cell being jumped too is unoccupied. The player cannot move or attack diagonally. A valid move is only horizontally or vertically. A move or an attack is not valid if there are no pieces on the board.

Things to Look Out For:

Throughout Yote capturing a piece is never mandatory. However, the player who successfully attacked has the option to attack again if the opportunity is present. In this case, the player removes one of the opponent's pieces for every successful jump in the attack made. If a situation where both players are down to three or less pieces, the game ends in a draw. Further, the game can end in a win for one player who has more pieces if their opponent has no available moves.

Yoté Rules

High Level Description of Yoté:

Yoté is concerned with the system as a whole, or larger components of it. The driver is connected to both the players and the board. It drives the program forward and keeps the turns going. The player has pieces both in your hand and on the board. The pieces are placed in cells that are part of the board. The board checks with the rules to see if the player's choices and turns are valid or not.

Goal:

The goal of the game is to capture all of the opponent's pieces, a player can capture when a piece is orthogonally adjacent to another, allowing you to jump to the empty cell immediately beyond it. The piece in the middle that was jumped over is removed from the board, followed by another piece of the player who successfully attacked their opponents choosing. First player to successfully attack all of their opponent's pieces, wins!

Board and Pieces:

The board to play Yoté is 5 cells by 6 cells large. One player has 12 white pieces and the other player has 12 black pieces with the white player going first. The players move, attack and place pieces throughout the board while alternating turns.

Valid Moves and Captures:

Moves are valid only when the jump is orthogonally adjacent to another piece. Placing a new piece on the board is valid if the cell is empty. A player can move around the board to a new position without attacking. However, if a player does choose to attack it is valid to have multiple captures in one move. When a player successfully attacks, they have the option to attack again if the opportunity is presented again. This is a strategy a player could use to make multiple attacks and capture more than one piece in a given move.

Use Cases

Use Case: Starting game
Primary Actor: Player
Goal: Initialize game to start condition
Stakeholders List: Player(s) want to play the game
Initiating Event: Both Players have indicated they want to start the game and are ready to start
Main Success Scenario:

1. The players will submit their choice of colour
2. The system will provided each player with 12 pieces of their colour
3. The system will generate the board, with all cells empty
4. The system will designate that it is the white player's Turn
5. The game beings

Post Conditions:
Both players have chosen their respective colours
Both players have 12 pieces of their colour
The board has been generated and is empty
The white players is ready to start their turn
Alternate Flows or Exceptions: N/A
Use Cases Utilized: N/A
Scenario Notes: N/A

Use Case: Turn
Primary Actor: Player
Goal: Allow the player to advance the game
Stakeholders List: Player wants to make a valid move, attack, or placement
Initiating Event: The system indicates that it is a specific player's turn
Main Success Scenario:

1. The player chooses a piece to perform a move, attack, or placement
2. The Player carries out their selected action (calls Single Attack, Move, or Placement)
3. The Player indicates their turn is over
4. The system checks with the Judge to see if a player has won (calls Win/Lose)
5. The judge indicates that no player has won
6. The system checks with the judge to see if the game is in a draw state (calls Draw)
7. The Judge indicates there is no draw
8. The system indicates the current player's turn is over and it is the next player's turn

Postconditions: The game has ended or it is the next player's turn
Alternate Flows or Exceptions:

2. The player decides they want to make an alternate action
3. The system directs the player to restart the current turn

-

5. The Judge indicates that a player has won
6. The system displays the winner

-

6. The Judge indicates that there is a draw
7. The system indicates that the game has ended in a draw

Use Cases Utilized: Move, Single Attack, Place, Draw, Win / Lose
Scenario Notes: N/A

Use Case: Single Attack
Primary Actor: Player
Goal: Attack enemy Player
Stakeholders List: Player wants to capture an enemy piece.
Initiating Event: It is the player's turn and they have chosen to make an attack
Main Success Scenario:

1. Player chooses which piece they would like to attack
2. The Judge checks whether this is a valid attack (calls Rule Check)
3. The Judge indicates the attack is valid
4. The system moves the piece to its new location on the board
5. The system designates the attacked piece for removal
6. The attack is concluded

Postconditions: The attacking piece is on its new square or the player is directed to restart their turn
Alternate Flows or Exceptions:

4. The Judge indicates the attack is not valid
5. The system displays the attack is not valid
6. The system directs the player to restart the current turn

-

7. The Player attempts to make an extended attack (Calls Extended Attack)

Use Cases Utilized: Extended Attack, Rule Check
Scenario Notes: N/A

Use Case: Extended Attack
Primary Actor: Player(s)
Goal: Continue an attack on the enemy player
Stakeholders List: Player wants to capture additional enemy piece(s)
Initiating Event: The Player has concluded a single attack
Main Success Scenario:

1. Player chooses which piece they would like to attack
2. The Judge checks whether this is a valid attack (calls Rule Check)
3. The Judge indicates the attack is valid
4. The system moves the piece to its new location on the board
5. The system designates the attacked piece for removal
6. The attack is concluded

Postconditions: The attacking piece is on its new square or the player is directed to restart their turn
Alternate Flows or Exceptions:

3. The Judge indicates the attack is not valid
4. The system displays the attack is not valid
5. The system directs the player to restart the current attack
   -
6. The Player attempts to make an extended attack (Calls Extended Attack)

Use Cases Utilized: Rule Check
Scenario Notes: N/A

Use Case: Take Piece(s)
Primary Actor: Player
Goal: To remove one or more attacked pieces from the board
Stakeholders List: Player wants to remove pieces they have captured
Initiating Event: A successful attack has been concluded
Main Success Scenario:

1. The system removes any pieces that were indicated for removal
2. The player indicates which additional enemy pieces they wish to remove
3. The system removes any additional pieces that were indicated for removal

Postconditions: All pieces indicated for removal have been removed from the board
Alternate Flows or Exceptions: N/A
Use Cases Utilized: N/A
Scenario Notes: N/A

Use Case: Move
Primary Actor: Player
Goal: To move a piece on the board to a new cell
Stakeholders List: Player(s) want to move their piece to a new location
Initiating Event: It is the player's turn and they have chosen to move a piece
Main Success Scenario:

1. Player chooses which cell on the board they are moving too
2. The judge checks whether the move is a valid move (calls Rule Check)
3. The judge indicates the move is valid
4. The system moves the piece to its new location on the board
5. The move is concluded

Postconditions: Player has moved the piece to a new cell successfully
Alternate Flows or Exceptions:

3. The judge has determined if the move is invalid
4. The system displays the move is not valid
5. The system directs the player to restart the current turn

Use Cases Utilized: Rule Check
Scenario Notes: N/A

Use Case: Placement
Primary Actor: Player
Goal: User places a new piece on an empty cell on the board
Stakeholders List: Player(s) wants to place a piece on the board
Initiating Event: It is the player's turn and they have chosen to place a piece

Main Success Scenario:

1. Player chooses which cell on the board placing a piece on
2. The judge checks whether the cell is empty (calls Rule Check)
3. The judge indicates the placement is valid
4. The system places the piece to its cell on the board
5. The placement is concluded

Postconditions: Player has placed a piece on a cell on the board successfully

Alternate Flows or Exceptions:

3. The judge has determined if the placement is invalid
4. The system displays the move is not valid
5. The system directs the player to restart the current turn

Use Cases Utilized: Rule Check

Scenario Notes: N/A

Use Case: Draw

Primary Actor: Judge

Goal: To determine if the game ends with no winner

Stakeholders List: Players want to know the outcome of the game

Initiating Event: Both players have three pieces remaining in the game.

Main Success Scenario:

1. The judge checks whether the draw conditions are met
2. The judge indicates that the game is in a draw state.

Postconditions: The game ended in a draw or the game continues.

Alternate Flows or Exceptions:

2. The judge indicates the game is not in a draw state

Use Cases Utilized: N/A

Scenario Notes: N/A

Use Case: Rule Check

Primary Actor: Judge

Goal: To ensure the rules are being followed

Stakeholders List: Player(s) want to know if their turn is valid

Initiating Event: A Player attempts to move, place or attack a piece

Main Success Scenario:

1. The judge checks whether the action violates the game rules
2. The judge indicates the game rules were not violated

Postconditions: The game is in a valid state

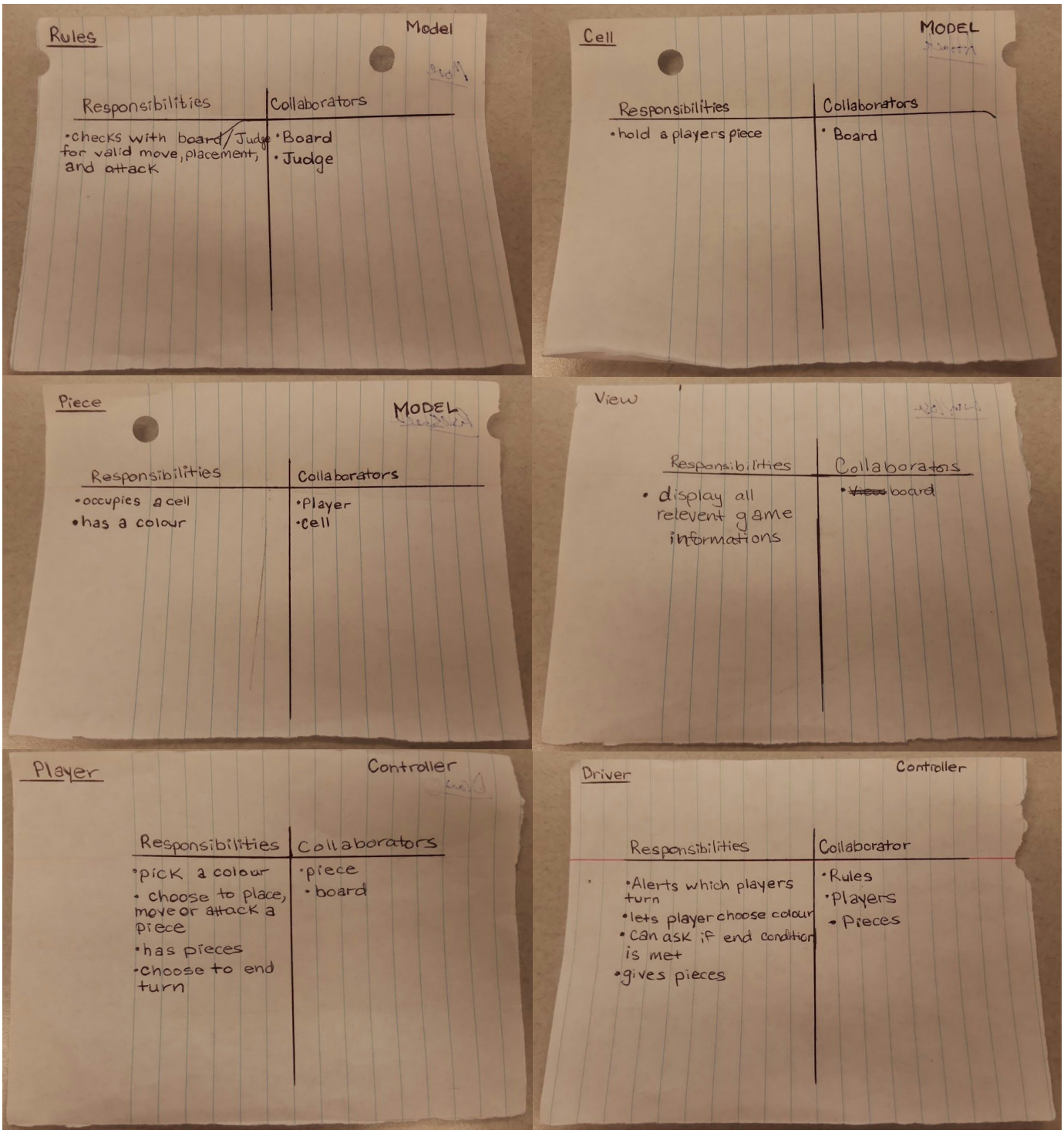
Alternate Flows or Exceptions:

2. The judge indicates the rules were violated

Use Cases Utilized: N/A

Scenario Notes: N/A

Reproduction of CRC Cards from the CRC session



Rules — Model

Responsibilities	Collaborators
• checks with board/Judge for valid move, placement, and attack	• Board • Judge

Cell — MODEL

Responsibilities	Collaborators
• hold a players piece	• Board

Piece — MODEL

Responsibilities	Collaborators
• occupies a cell • has a colour	• Player • Cell

View

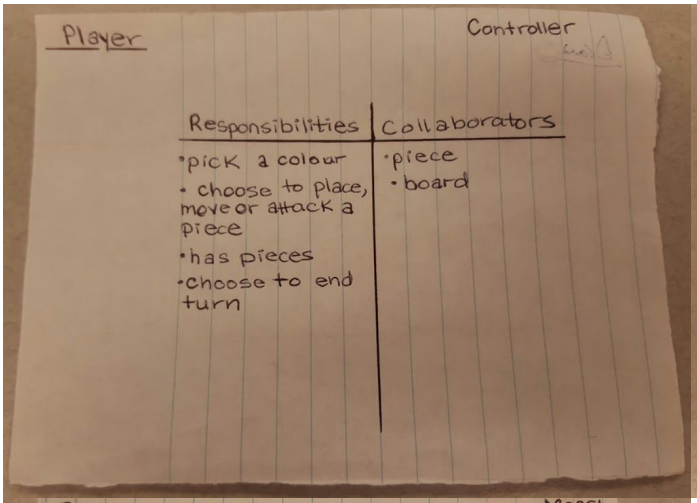
Responsibilities	Collaborators
• display all relevent game informations	• ~~View~~ board

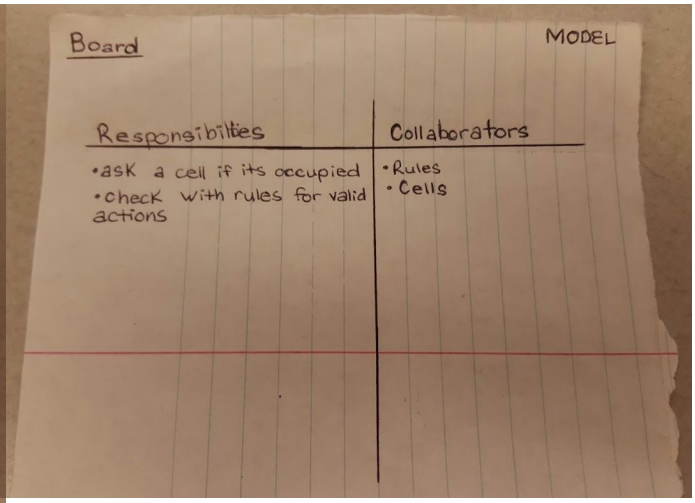
Player — Controller

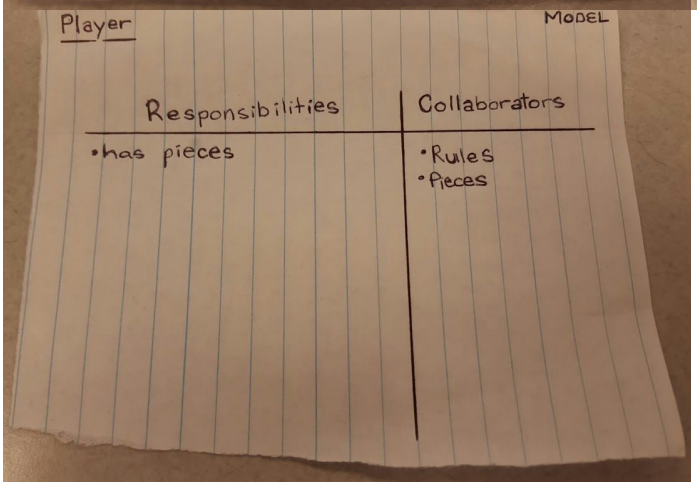
Responsibilities	Collaborators
• pick a colour • choose to place, move or attack a piece • has pieces • choose to end turn	• piece • board

Driver — Controller

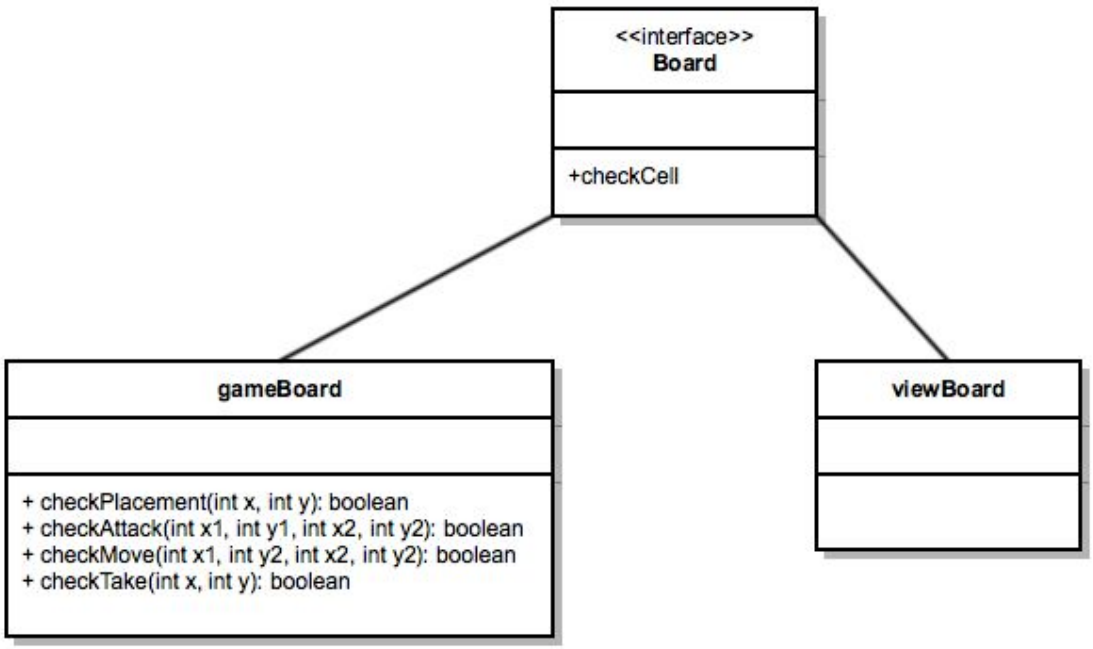
Responsibilities	Collaborator
• Alerts which players turn • lets player choose colour • can ask if end condition is met • gives pieces	• Rules • Players • Pieces


Player
Controller
Responsibilities
Collaborators
•pick a colour
• choose to place, move or attack a piece
•has pieces
•choose to end turn
•piece
• board

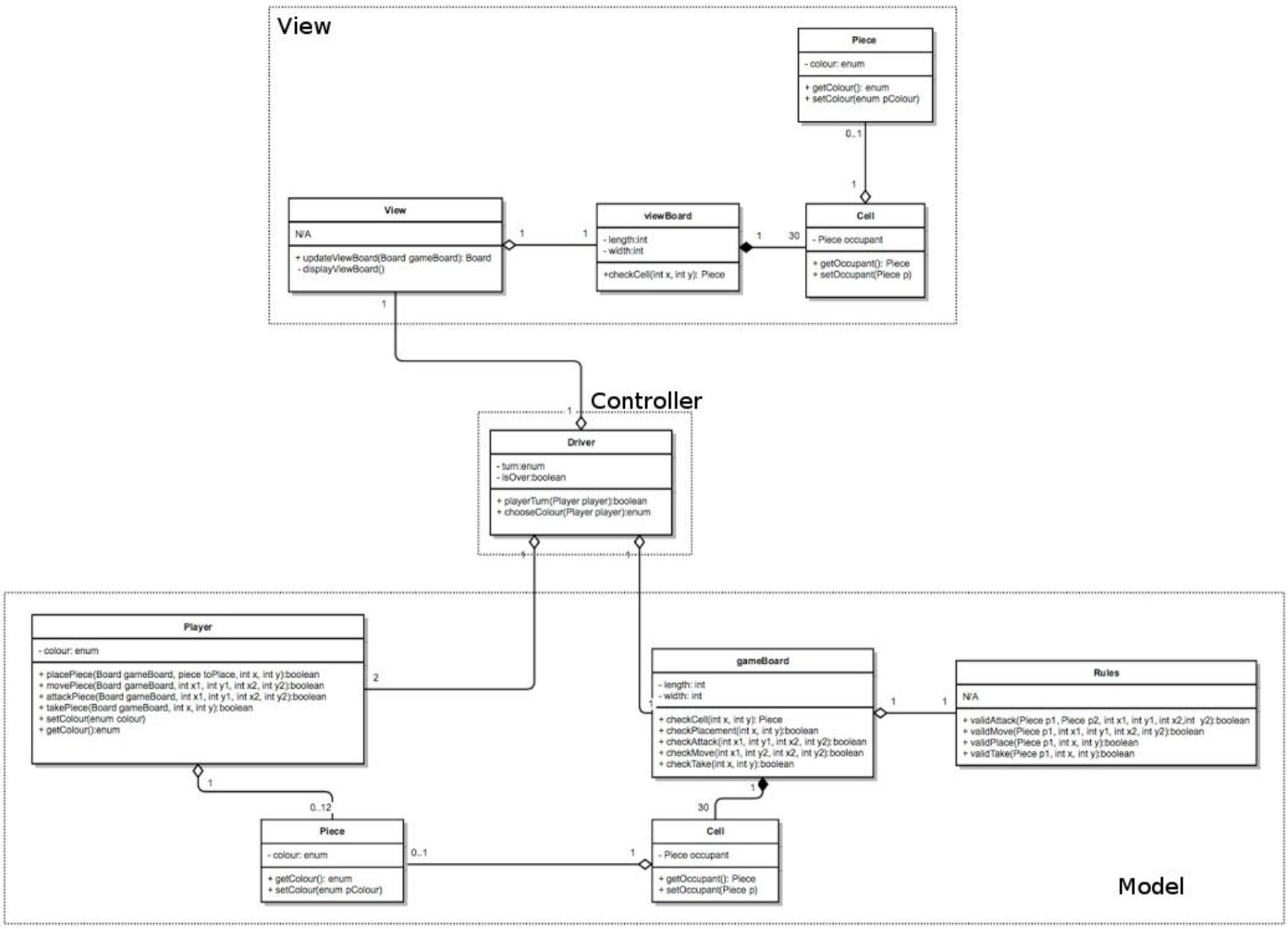

Board
MODEL
Responsibilities
Collaborators
•ask a cell if its occupied
•check with rules for valid actions
•Rules
• Cells


Player
MODEL
Responsibilities
Collaborators
•has pieces
•Rules
•Pieces

Interface/Type model



Class Diagram



Class Information

Piece

Constructors:
Piece()
Piece(enum colour)
Instance Variables:
enum colour
Methods:
public enum getColour(): Returns the colour of the piece

public void setColour(enum pColour): Sets the colour of the piece

---

gameBoard

Constructors:
Board()
Board(int length, int width)
Instance Variables:
int length
int width
Methods:
public Piece checkCell(int x, int y): Checks a cell to see if it is occupied, returns either NULL or the piece occupying the cell.

public boolean checkPlacement(int x, int y): Checks the cell the player is trying to place their piece on, then uses the rules class to check if the placement is valid.

public boolean checkAttack(int x1, int y1, int x2, int y2): Checks the cell the player is attacking from and the cell the player is attacking, then uses the rules class to check if the attack is valid.

public boolean checkMove(int x1, int y2, int x2, int y2): Checks the cell the player is moving from and the cell the player is moving to, then uses the rules class to check if the attack is valid.

public boolean checkTake(int x, int y): Checks the cell player is trying to take the piece from, then uses the rules class to check if the removal is valid.

Cell

Constructors:
Cell()
Cell(Piece occupant)
Instance Variables:
Piece occupant
Methods:
public Piece getOccupant(): Returns the piece in the cell

public void setOccupant(Piece p): Sets the piece in the cell

Player

Constructors:
Player()
Player(enum colour)
Instance Variables:
enum colour
Methods:
public boolean placePiece(Board gameBoard, piece toPlace, int x, int y): Using gameBoard call checkPlacment to see if the placement is valid, if return is true, sets the piece to the cell. Returns true or false.

public boolean movePiece(Board gameBoard, int x1, int y1, int x2, int y2): Using gameBoard call checkMove to see if the move is valid, if return is true, sets the piece to the cell. Returns true or false.

public boolean attackPiece(Board gameBoard, int x1, int y1, int x2, int y2): Using gameBoard call checkAttack to see if the attack is valid, if return is true, sets the piece to the cell. Calls takePiece to take remove pieces from the board. Returns true or false.

public boolean takePiece(Board gameBoard, int x, int y): Allows the player to choose what pieces they would like to remove and removes any pieces that have been attacked. Returns true or false

public void setColour(enum colour): Sets the colour of the player

public enum getColour(): Returns the player's colour

Rules

Constructors:
Rules()
Instance Variables:
N/A
Methods:
public boolean validAttack(Piece p1, Piece p2, int x1, int y1, int x2, int y2): Checks whether the attack is valid, returns true or false.

public boolean vaildMove(Piece p1, int x1, int y1, int x2, int y2): Checks whether the move is valid, return true or false

public boolean validPlace(Piece p1, int x, int y): Checks whether the placement is valid, returns true or false

public boolean validTake(Piece p1, int x, int y): Checks whether the removal is valid, returns true or false

---

View

Constructors:
View()
Instance Variables:
n/a
Methods:

public Board updateViewBoard(Board gameBoard): Updates the view board with the new state of the gameBoard.

private void displayViewBoard(): Displays the board on the screen

---

viewBoard

Constructors:
viewBoard()

viewBoard(int length, int width)

Instance Variables:

int length

int width

Methods:

public Piece checkCell(int x, int y): Checks a cell to see if it is occupied, returns either NULL or the piece occupying the cell.

Driver

Constructors:

Driver()

Instance Variables:

enum turn

boolean isOver

Methods:

public boolean playerTurn(Player player): Lets player know it is their turn.

public enum chooseColour(Player player): Allows players to choose their colour, returns the colour that was chosen.
