CIS 17C Project 2

<Mancala>

CIS 17C-43484
Paul Ingram
06/11/2022

Introduction

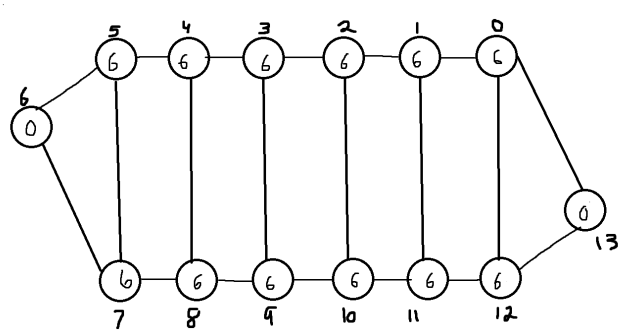
What are you coding and why did you choose this game?
For this project I decided to code Mancala. I chose this game because I used to play it all the time when I was younger. I even had a Club Penguin themed mancala board. Another reason is because both of my 17A projects and my first 17C project were Mancala and I wanted to see how I would code it differently this time. I think Mancala has simple enough rules that provide a clear plan for developing the code.

How long did you spend, how many lines, classes, etc.?
I spent about a week coding this game. I finished with about 1,138 lines. The project uses 4 classes where each class serves as a different game mode with one class holding the information for each player. There is a player vs. player, player vs bot, and bot vs bot gamemode where the bot vs bot gamemode collects statistics of the frequency each cell is played. The bot vs bot mode also runs 1000 games and counts how many wins each player gets. I developed a system which recommends a cell to play so when I pitted two bots who played the recommended cells against each other I found that whoever has the first turn will always win. I then changed the simulated game to be a recommended move vs random move bot that always selects a cell with at least one marble in it and got different statistics. However, I still found that the recommended moves win a majority of the games.

Where on GitHub is it located?
https://github.com/pingram32/IngramPaul_CIS17C_43484

Approach to Development

Concepts
I first started with creating a graph (pictured below) that I could reference throughout coding Mancala.



Each node of the graph represents a cell that has two neighbors, the adjacent node and the node that was across the board from it. I needed to know which node was across from each cell so I could implement one of the rules of Mancala where if a marble ends on an empty cell on the player's side they capture the cell across from it. I created the graph using an adjacency list. After developing the gameboard I then made a player class which contains hashing functions for the name the user inputs. I used a simple polynomial rolling hash function which involved multiplication and addition. I also made a recursive function which updates the board based on what cell the player chooses. The recursive function returns the last cell of the play and determines if there is a capture or a go again function that must be executed. Like I previously mentioned, I developed a function which recommends a cell to play. The function uses a recursive sort to sort the cells on the player's side and recommends a cell based on which cells are returned from the sort. I used 2 different recursive sorts here with a selection sort and a bubble sort. Lastly, I used a recursive merge sort for the bot vs bot game mode in order to count the frequency each cell was played.

Version Control

V1 Make the graph with an adjacency list and keep track of the initial marbles with a vector. Develop player one's move and fill the adjacency list.

V2 Put everything in a class. Make player two's turn.

V3 Make a player class. Develop a hash function for player strings.

V4 Make a recursive function for a player's turn.

V5 Make a recommended move function for player 1 that uses a recursive bubble sort.

V6 Same as **V5** but created the function for player 2. Still uses recursive bubble sort.

V7 Make a bot v bot class and make a recursive merge sort.

V8 Use the merge sort to count the frequency of cell plays. Class also returns the total plays, wins, and ties.

V9 Make a player vs bot gamemode and use a selection sort instead for the recommended move function. The bot plays a random cell with at least one marble in it each play.

Game Rules

The rules are simple in Mancala. Each side has 6 pits where each pit starts off with 4 marbles. The first player chooses a pit and adds one marble to each of the following pits until they no longer have any marbles left from the pit that they chose from. If the player happens to land their last marble in their score pit (located to the right and left of each side) they receive another turn. If the last marble lands in an empty pit on the same side it was played from the pit across from it is captured and goes to that side's score bank. Once the first player has finished making their move, the second player chooses a cell on their respective side. Neither player may select a cell that is not on their side. The game ends once one side has 0 marbles left in all of the pits. Once the game is finished all the marbles left over are consolidated to their respective side's score pit and the winner is determined by which player has more marbles in their score pit.

Description of Code

Organization
The project is organized into a main file, a BotGame class, a GameBoard class, a Player class, and a Simulate class. The player chooses in main which game mode they would like to play and then the constructor is created for whichever mode they chose.

Classes
BotGame, GameBoard, Player, Simulate

Sample Input/Output

```
Main Menu...
1) 1v1
2) 1vBot
3) Simulate for statistics
```

```
This is Mancala
Enter Player 1 Name: Paul
Enter Player 2 Name: Lehr
|6|6|6|6|6|6|
0==========0
|6|6|6|6|6|6|
Lehr select a cell on the bottom row 1-6 (L-R)
The recommended cell is 1
```

```
You have earned another turn, go again
|6|6|6|6|6|6|
0==========1
|0|7|7|7|7|7|
```

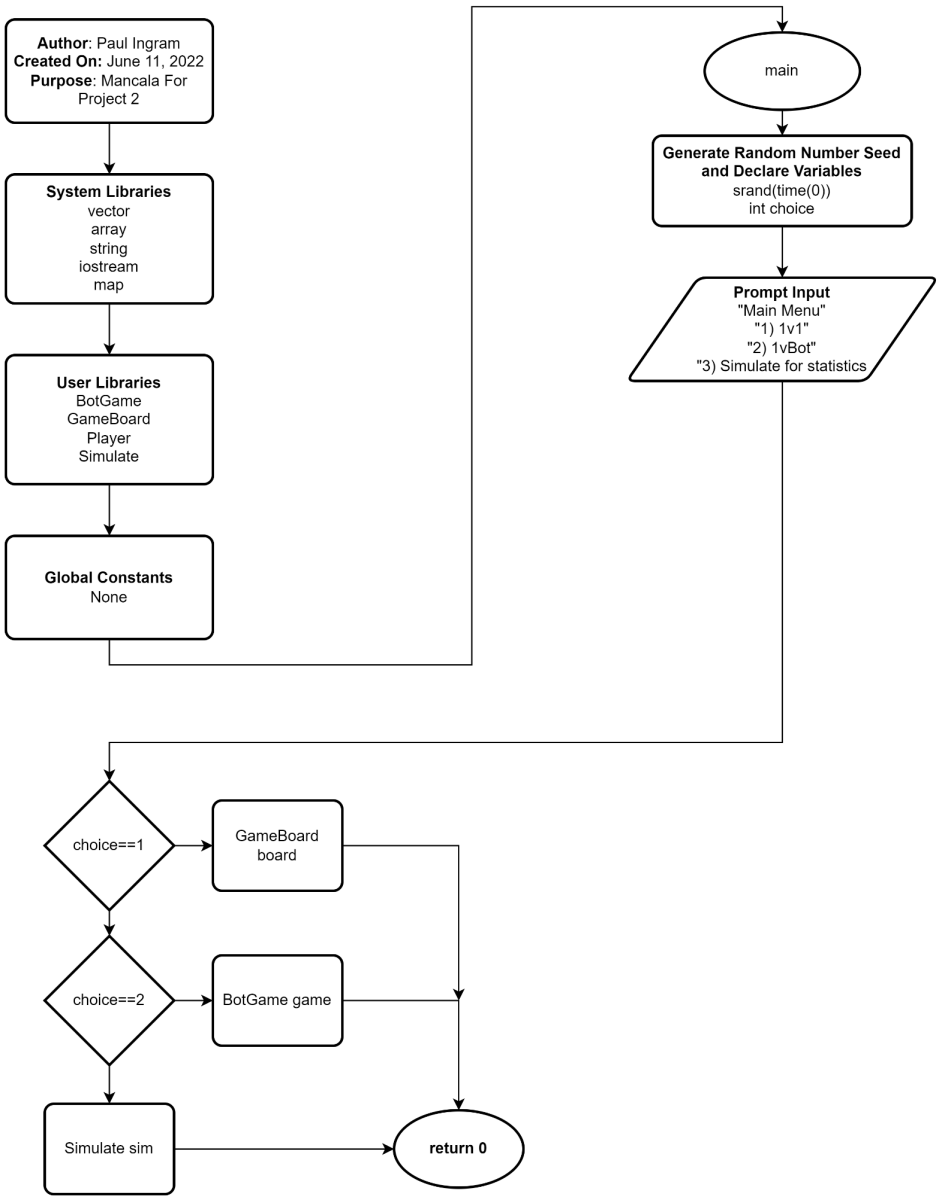
```
Mancala Statistics Version
Player One Total Plays:
Cell 1 Plays: 12708
Cell 2 Plays: 11343
Cell 3 Plays: 7864
Cell 4 Plays: 4488
Cell 5 Plays: 2422
Cell 6 Plays: 2602
Player Two Total Plays:
Cell 1 Plays: 4892
Cell 2 Plays: 4784
Cell 3 Plays: 4917
Cell 4 Plays: 4838
Cell 5 Plays: 4847
Cell 6 Plays: 4731
Player 1    Wins: 950
Player 2    Wins: 41
Ties            : 9
```

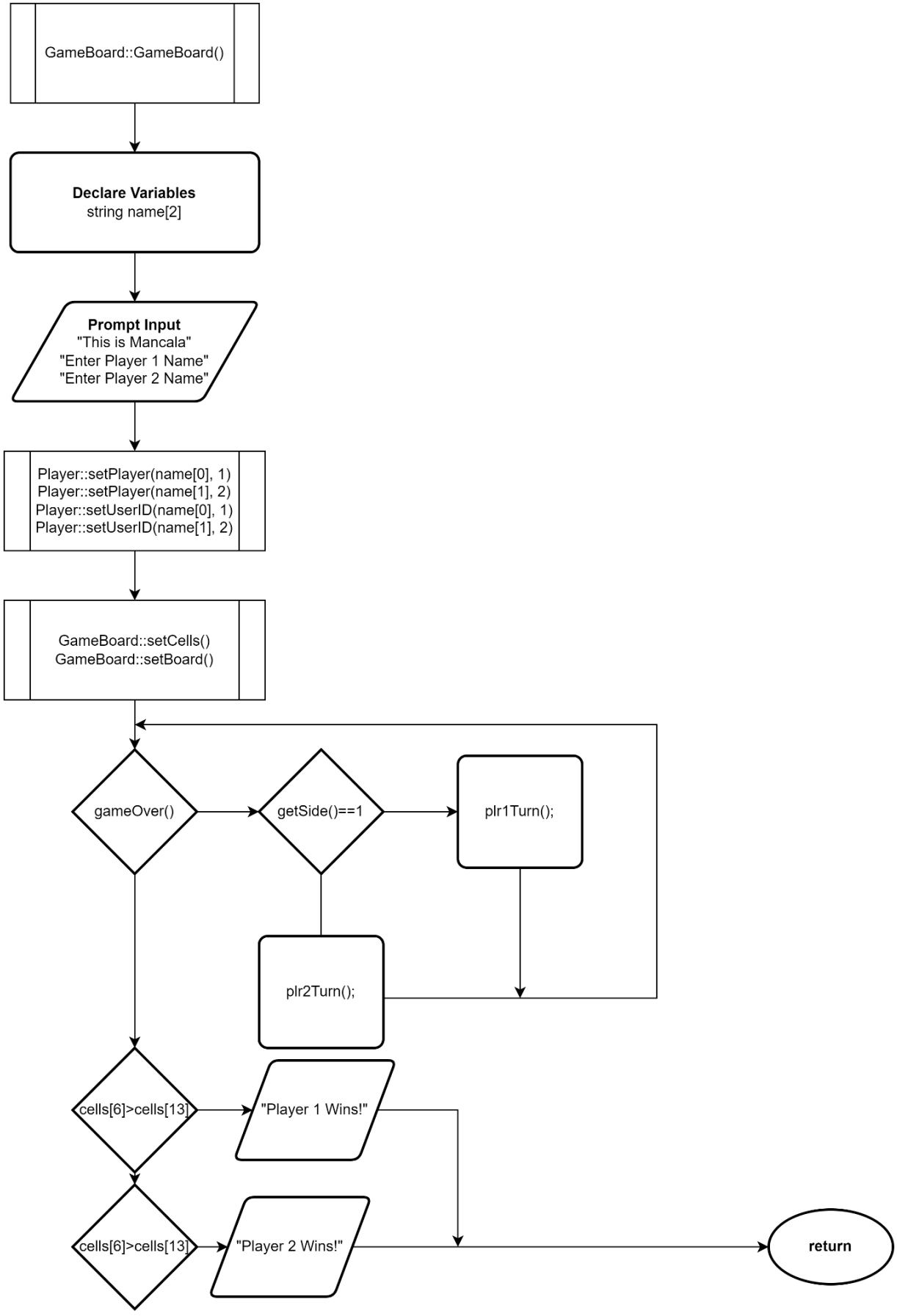
Checkoff Sheet

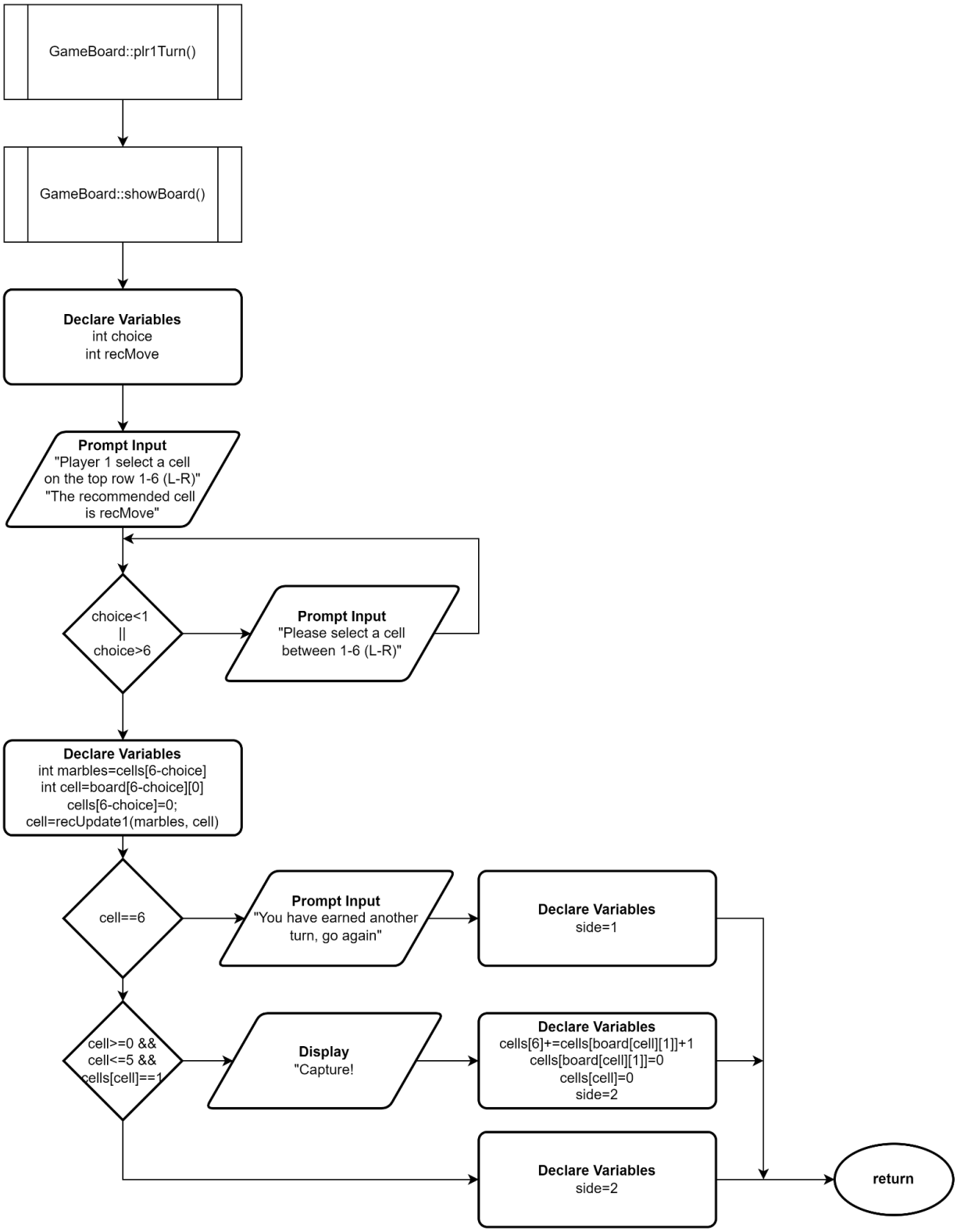
Recursions GameBoard::Line 117
Recursive Sorts
- **Bubble Sort** GameBoard::Line 163
- **Merge Sort** Simulate::Line 228
- **Selection Sort** BotGame::Line 118

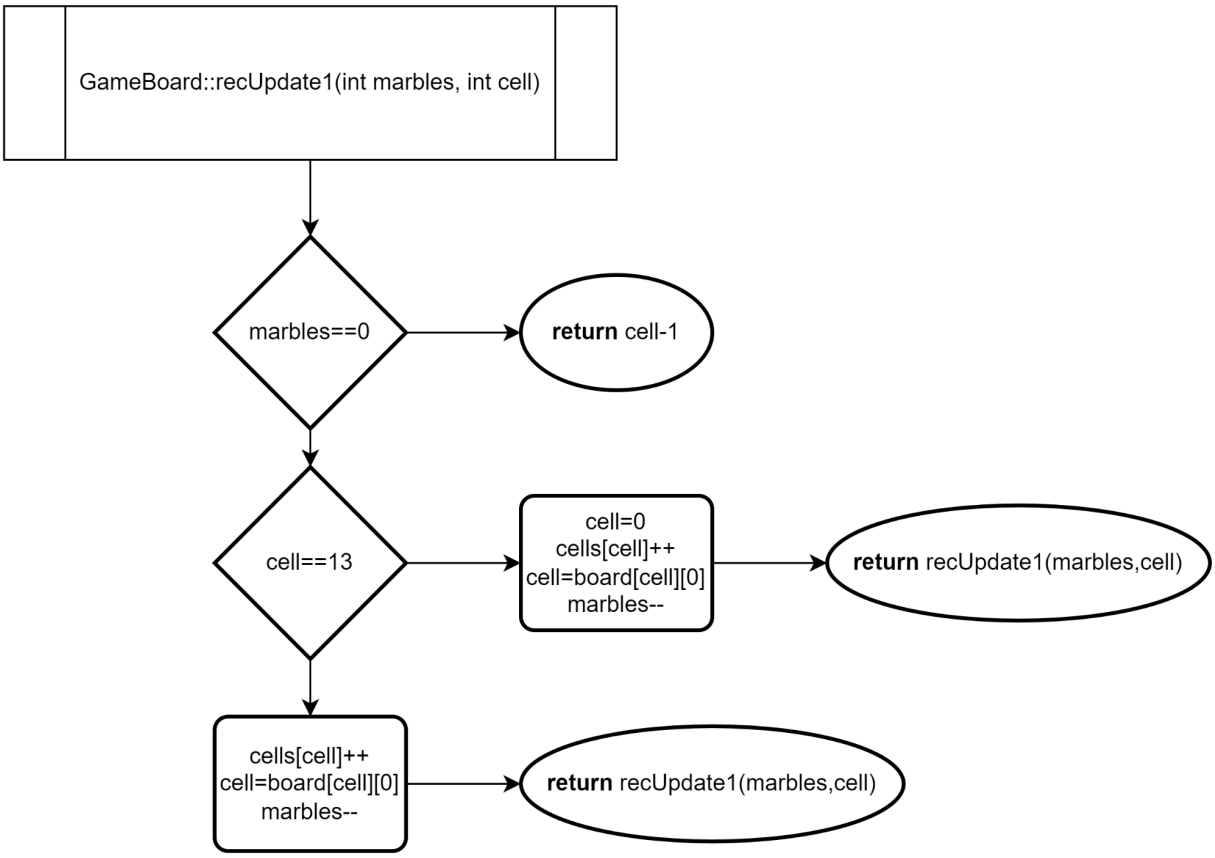
Hashing Player::Line 28
Graphs/Trees GameBoard:: Line 265

Flowchart




GameBoard::GameBoard()
Declare Variables
string name[2]
Prompt Input
"This is Mancala"
"Enter Player 1 Name"
"Enter Player 2 Name"
Player::setPlayer(name[0], 1)
Player::setPlayer(name[1], 2)
Player::setUserID(name[0], 1)
Player::setUserID(name[1], 2)
GameBoard::setCells()
GameBoard::setBoard()
gameOver()
getSide()==1
plr1Turn();
plr2Turn();
cells[6]>cells[13]
"Player 1 Wins!"
cells[6]>cells[13]
"Player 2 Wins!"
return


GameBoard::plr1Turn()
GameBoard::showBoard()
Declare Variables
int choice
int recMove
Prompt Input
"Player 1 select a cell
on the top row 1-6 (L-R)"
"The recommended cell
is recMove"
choice<1
||
choice>6
Prompt Input
"Please select a cell
between 1-6 (L-R)"
Declare Variables
int marbles=cells[6-choice]
int cell=board[6-choice][0]
cells[6-choice]=0;
cell=recUpdate1(marbles, cell)
cell==6
Prompt Input
"You have earned another
turn, go again"
Declare Variables
side=1
cell>=0 &&
cell<=5 &&
cells[cell]==1
Display
"Capture!
Declare Variables
cells[6]+=cells[board[cell][1]]+1
cells[board[cell][1]]=0
cells[cell]=0
side=2
Declare Variables
side=2
return



Pseudocode

Get game mode from user
If user selects 1v1 get both user names and construct a Board class
Set the current player to a random number between 0 and 1
While the game is not over
    If the current player is equal to 0 gather play from the user
    Update the game history
    Update the game board with the user's choice while checking for second chances and captures
    Set the current player to 1
    If the current player is equal to 1 gather play from the user
    Update the game history
    Update the game board with the user's choice while checking for second chances and captures
    Set the current player to 1
Once the game is over announce the winner and their score

Else gather user name and construct a SimGame class
Do the same as if it was a 1v1 but generate a random number for the bot player

UML Diagram