Connect Four Design Documents

Plukov, George
plukovga@mcmaster.ca

Stone, Theo
stonet@mcmaster.ca

Habib, Philip
habibp@mcmaster.ca

White, Carson
whitec7@mcmaster.ca

Odd, Tim
oddt@mcmaster.ca

April 6, 2015

Part I

CLASSES

CONTENTS

1

CONNECT FOUR

1.1 Description

This class initializes the display screen and the graphics library called lwjgl (light weight java game library). these two things create the window and the method to draw on the window. It contains the game loop and when the game is over cleans up the display and the mouse links.

1.2 Interface

1.2.1 Uses

NONE

1.2.2 Types

NONE

1.2.3 Code Breakdown

public static void main(String [] args) throws IOException
Initializes the display, openGL, and our game board. Also begin's the game loop and clean up after it finishes.

1.3 Implementation

1.3.1 Uses

org.lwjgl.opengl.GL11.GL_COLOR_BUFFER_BIT
org.lwjgl.opengl.GL11.glClear
org.lwjgl.opengl.GL11.glLoadIdentity
java.io.IOException
org.lwjgl.LWJGLException
org.lwjgl.input.Mouse

org.lwjgl.opengl.Display
org.lwjgl.opengl.DisplayMode
org.lwjgl.opengl.GL11

1.3.2 Types

int WIDTH
int HEIGHT
String GAMENAME
GameBoard game

1.3.3 Code Breakdown

public static void main(String [] args) throws IOException This method calls our basic startup methods for the game. It prepares the OpenGL, initializes the game, starts the game loop, and performs the cleanup after the game is finished.

```
public main
    initialize GL
    initialize Game
    start gameloop
    perform cleanup
```

private static void initDisplay()

```
private initDisplay
    set display mode width and height
    create display and mouse
    set title
    enable vsync
```

private static void initGL()

```
private initGL
    intializes OpenGL for 2d graphics
```

private static void initGame() throws IOException

```
private initGame
    create gameboard
```

private static void gameLoop()

```
Private gameLoop
    while game is not closed
        update
        render
```

private static void update()

```
private update
    update game state
```

private static void render()

```
private render
    reset matrix for frame
    update display
    set frame rate to 60
```

private static void cleanUp()

```
private cleanUp
    destroy mouse
    destroy display
```

2

GAME BOARD

2.1 Description

This class holds the current state of the game. It holds the current state of the chips and the states of any invalid move pieces. It receives information from the input class(to update the state) and sends information to the View class(to draw the model). This is the model.

2.2 Interface

2.2.1 Uses

NONE

2.2.2 Types

NONE

2.2.3 Code Breakdown

public GameBoard()

```
public GameBoard()
    initialize chips array
    initialize errors array

    if (random boolean)
        red player first
    else
        blue player
```

Initialize a new game board with a 7x6 chip grid. Also selects which player will play the first move. This initiallizes our model, and selects its starting state.

public void update()

```
public void update()
```

```
check isRed for current color
update mousPos with current mouse position
for chip rows
    for chip columns
        if mouse postion is in range
            if isRed is true
                place red chip
                else
                place blue chip
```

Get the input from the view(MouseInput) and update the model(Chips array). We chose to use two arrays, one to keep track of the chips on the board. The other keeps track of the empty spaces beneath a chip which is causing it to be an invalid move. For example, if a chip is floating in the air the empty spaces under it will be highlighted.

public boolean isRedActive()
Returns whether or not it is the red players turn to play. Used to add what color should be stored in the chip variables in the chip array.

public Chip [][] getChips()
Returns the Chips array to allow the View to draw them to the screen.

2.3 Implementation

2.3.1 Uses

org.newdawn.slick.geom.Point
java.io.BufferedReader
java.io.FileNotFoundException
java.io.FileReader
java.io.PrintWriter
engine.ChipColor.Color

2.3.2 Types

boolean isRed
Chip [][] chips

2.3.3 Code Breakdown

private static boolean addChip(Chip [][] chips, int column, Color c)

```
private static boolean addChip(Chip [][] chips, int column, Color c)
    for each column:
```

```
        if clicked spot is empty:
            add a new chip to that spot
            return true

    if spot is full
        return false
```

	returns
Chips[col clicked][y] == null	true
Chips[col clicked][y] != null	false

public boolean isRedActive()
Get the current players turn

public Chip[][] getChips()
Get the current state of the chips on the board

public void saveGame() This method is used to save the current game state to a text file. This includes the positions of the chips on the bored and the current players turn.

```
private void saveGame()
    open save.txt
    write current players turn

    for each chip
        if chip color == null
            write "_" to save.txt
        if chip color == blue
            write "b" to save.txt
        if chip color = red
            write "r" to save.txt
        write new line to save.txt
    close file
```

private void loadGame() Load the game from the save.txt file. Load each b, r, or null into the chips array as a blue, red or no chip.

```
private void saveGame()
    open save.txt for reading

    check whose turn it is

    for each item in the file:
        check if its blue or red or null and add it to the chips list
```

3

CHIP

3.1 Description

This clas represents a single chip. It has a property to hold the color(which player played it). It also has properties to determine where it should be drawn.

3.2 Interface

3.2.1 Uses

NONE

3.2.2 Types

NONE

3.2.3 Code Breakdown

public Chip(int x, int y, Color c)
Takes the coordinate of input and the color of a chip and stores it for future reference.

public ChipColor getColor()
Returns the color of the chip.

public int getX()
Returns the x coordinate of the chip for use in drawing.

public int getY()
Returns the y coordinate of the chip for use in drawing.

3.3 Implementation

3.3.1 Uses

engine.ChipColor.Color

3.3.2 Types

int x
int y
Color color

3.3.3 Code Breakdown

No private methods. Since this is a data structure no calculations occur in the class.

4

LOGIC

4.1 Description

This class checks if the current state of the game board is a valid state. Sepcifically, it checks to make sure there are no floating chips and that there are an appropriate number of blue chips relative to the number of reds.

4.2 Interface

4.2.1 Uses

NONE

4.2.2 Types

NONE

4.2.3 Code Breakdown

public static Color checkWin(Chip [][] chips)

	Color
if c != null	c
c == null	null

Determines whether or not to add the chip to the array.

public static boolean isTie(Chip [][] chips)

	return
Chips[x][y] == null	true
Chips[x][y] != null	false

4.3 Implementation

4.3.1 Uses

NONE

4.3.2 Types

NONE

4.3.3 Code Breakdown

private static Color check(Chip [][] chips, int x, int y)

```
public Color check(Chip [][] chips, int x, int y):
    for each column:
       for each row:
            check if that chip is part of a win
            if !null
                return c

    return null
```

private static boolean linearMatch(Chip [][] chips, int x, int y, int stepX, int stepY)
This method is used to check if there is a victory. It checks in the direction of stepx and stepy. An example of this would be stepx= 1 stepy = 0, this would check the line of 4 items to the right of the current chip. It returns false if there is no line of 4 items, and returns true if the current chip is involved in a win.

```
private static boolean linearMatch(Chip [][] chips, int x, int y, int stepX, int s
    chipColor = chips[x][y]
if (any chips in range of stepX and stepY.color != chipColor)
return false
else
return true
```

public static boolean isTie(Chip [][] chips)
This method is used to check if the gameboard is tied. This is done by seeing if the gameboard is totally full and no other pieces can be added. Since we are checking for a win after a piece is placed, the game must be a tie if this method returns true.

```
public static boolean isTie(Chip [][] chips):
    for each column
```

```
    for each row
        if no chip
            return false
if board is full
    return true
```

5

MOUSE INPUT

5.1 Description

This module hooks into lwjgl to access the mouse position. It is also used to determine if a click has occured.

5.2 Interface

5.2.1 Uses

NONE

5.2.2 Types

NONE

5.2.3 Code Breakdown

public static Point getMousePosition()

```
public getMousePosition
    return the x and y position of the mouse
```

This method will return the position of the mouse so that we can check whether it is a correct move, and then add it to the board. This is a very important method as it is the main part of playing the game. It allows the player to place items on the board.

public static boolean isClicked()

```
public isButtonDown
    returns true if the right mouse button is clicked
```

Is used in conjunction with getMousePosition() to get a click. The value that is passed into this determines which button should be clicked, in this case 0 for left click. This method was made to create the seperation need in an MVC setup and is referenced statically.

5.3 Implementation

5.3.1 Uses

org.lwjgl.input.Mouse org.newdawn.slick.geom.Point

5.3.2 Types

NONE

5.3.3 Code Breakdown

This module had no private methods. Since all of the methods were mostly links into the lwjgl there was no need to have any helper methods, and since this is a module the only private methods needed would be the helper methods.

6

VIEW

6.1 Description

This class is what dispalys the graphics on the Display. It creates the SpriteSheet and renders the GameBoard with the Connect 4 grid and draws the chips.

6.2 Interface

6.2.1 Uses

GameBoard SpriteSheet

6.2.2 Types

NONE

6.2.3 Code Breakdown

public static void render(GameBoard game)

```
public render

    draw chips on bottom corners to select color to play

    draw save button
    draw load button
    draw reset button

    draw the outline of the selected chip
    chips [][] = getChipArrayFromGameBoard
    for i = 0 to chips length
        for j = 0 to chips[0] length
            if (chips[i][j] is not null)
                draw Chip to screen
```

```
    if (logic.checkValidity())
        draw "Valid"
    else
        draw "Invalid"

    for i = 0 to chip length
        for j = 0 to chip[0] length
            draw grid x
            draw grid y
if error
                highlight error
```

Renders the GameBoard with tiles

6.3 Implementation

6.3.1 Uses

engine.ChipColor.Color

6.3.2 Types

SpriteSheet

6.3.3 Code Breakdown

private static void drawChip(Chip chip)

```
private drawChip
    switch on public enum
        case BLUE
            draw blue chip on clicked position
        case RED
            draw red chip on clicked position
```

Draws a red or blue chip on the clicked position

7

SPRITE SHEET

7.1 Description

This class uses a relative path to a PNG image and splits it into tiles based on a grid. IT also provides some drawing support.

7.2 Interface

7.2.1 Uses

NONE

7.2.2 Types

NONE

7.2.3 Code Breakdown

public SpriteSheet(String address, float imgX, float imgY)
This initializes our sprite sheet. The parameters taken in are the width and height of the tile in the sprite sheet. It also takens in the path to where the sprite sheet image is saved.

public void draw(float x, float y, float gridX, float gridY)
Draws the sprite to the screen between the given bounds. We decided on this method because it would allow us to expand our program and use animations if we wanted too.
public int getXSize()
Returns the width of the sprite.
public int getYSize()
Returns the height of the sprite.

7.3 Implementation

7.3.1 Uses

NONE

7.3.2 Types

Texture texture
float imgX
float imgY

7.3.3 Code Breakdown

This class didn't need any private methods.The private variables were used to hide the irrelevant information from potentially being manipulated.

8

ARTIFICIAL INTELLIGENCE

8.1 Description

This module is used to control the artificial intelligence. It contains methods to check wins, and eventually to decide on the column to put the second players piece. The artificial intelligence always plays using the blue pieces. This module is part of our controller, it gives the model a value that will change the state of the game.

8.2 Interface

8.2.1 Uses

NONE

8.2.2 Types

NONE

8.2.3 Code Breakdown

public static int getMove(Chip[][] chips)
This method returns the column in which the AI's move should be made. It looks at the model(current state of the game) and makes a decision based on the moves made up to this point. This is part of the controller, similar to the Logic module the Artificial Intelligence module is only statically refferenced. This method returns the column that the AI has decided to place a chip in.

8.3 Implementation

8.3.1 Uses

java.util.ArrayList;
engine.ChipColor.Color;

8.3.2 Types

NONE

8.3.3 Code Breakdown

public static int getMove(Chip[][] chips) This method uses the other internal methods in the Artificial intelligence module to find the move that the AI should make. It checks whether the AI can win in the next turn or whether the player can win in the next turn and then bases its move off of that decision.

```
public static int getMove (chip[][] chips):
    if AI cant win:
        return the chosen column

    if there is no middle chip:
        return the middle column

    if the center isnt full:
        return a random column

    if the edges aren't full:
        return an edge (0 or 6)

    return a random edge
```

private static int checkAIWin(Chip[][] chips) This method is used to check whether or not the AI can win on the next move. It returns the column the AI should play if there is a possible winning move.

```
private static int checkAIWin(Chip[][] chips):
    if there are three pieces in a horizontal line:
        return the block/win move

    if there are three pieces ina vertical line:
        return the block/win move

    if there are three pieces in a row upward diagonals:
```

```
        return the block/winning move

    if there are three pieces in a row downward diagonals:
        return the block/winning move

    otherwise return -1
```

private static int checkPlayerWin(Chip[][] chips) This method checks whether or not the player can win on the next turn. If the player has a winning move possible the AI will fill the spot to block it.

```
private static int checkPlayerWin(Chip[][] chips):
    if player horizontal win:
        return blocking move

    if player vertical win:
        return blocking move

    if player diagonal upwards win:
        return blocking move

    if player diagonal downwards win:
        return blocking move
```

private static int checkHorizontal(Chip[][] chips, Color c) This method is used to check whether there is three of the same color in a row. The computer uses this information to make a move that will block the players win.

```
private static int checkHorizontal(Chip[][] chips, Color c):
    for all chips:
        if chips beside are same color
            return col

    return -1
```

textbprivate static int colorChecker(ArrayList¡Color¿ cList, Color c) THis method is used to check if there are three chips of the same color in a line. It will return the position of the space to move, if no spots exist then it will return -1.

```
private static int colorChecker(ArrayList<Color> cList, Color c):
    if clist contains c:
        return -1
```

```
    if clist doesnt contain null:
        return -1
    else:
        for color in clist
            if c == color
                colorCount ++
        if colorCount == 3s
            return where clist is null
        else:
            return -1
```

private static int checkVertical(Chip[][] chips, Color c) This method is used to see whether there is 3 chips of any color in a row. The color doesnt matter because the AI will want to make the blocking move (three reds) or the winning move (3 blues) no matter what.

```
private static int checkVertical(Chip[][] chips, Color c):
    for each chip on the gameboard:
        if the chip above and the chip below are the same color:
            return the blocking position
    return -1
```

private static int checkDiagonalUp(Chip[][] chips, Color c) This method is used to check if there are three of the same color piece going in an upwards diagonal from the chip. This is done in a similar way to checkHorizontal, by looking at the pieces around that chip and seeing if there is a winning or blocking move.

```
private static int checkDiagonalUp(Chip[][] chips, Color c):
    for each chip on the gameboard:
        add the four upward diagonal pieces to an arraylist
        check if there is a possible win/block
    return -1 for no diagonal win/block
```

private static int checkDiagonalDown(Chip[][] chips, Color c) This method is the exact same as checkDiagonalUp except it checks the lower end of the board. It looks at the diagonally surrounding pieces to see if they are the same color as the current piece. Then it will return the column to make a move or -1 if there is no diagonal winning move.

```
private static int checkDiagonalDown(Chip[][] chips, Color c):
private static int checkDiagonalUp(Chip[][] chips, Color c):
    for each chip on the gameboard:
        add the four downward diagonal pieces to an arraylist
        if there is a possible win/block
```

```
            return column of win/block
    return -1 for no diagonal win/block
```

private static boolean centerFull(Chip[][] chips) This method checks to see if the center of the board is full by checking if the tops of the center columns are null. If all of the chips at [i][5] are null then the center cannot be full.

```
private static boolean centerFull(Chip[][] chips):
    for all chips in the top row
        if that chip is null:
            return false
    return true
```

9

CONNECT FOUR TESTING

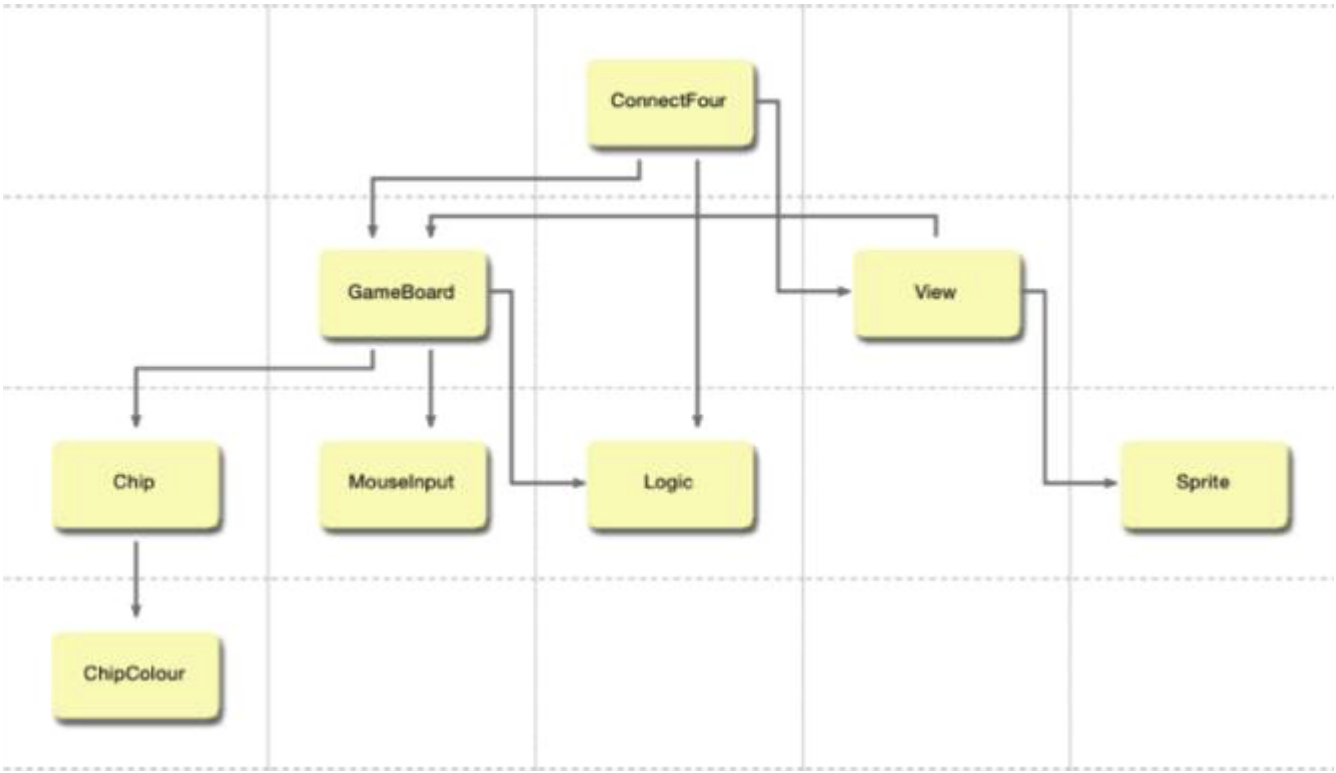
Mouse Input:
- Tested that nothing happens to the view when clicking outside of the board.
- Tested that when selecting the blue tile and clicking the grid makes a blue tile appear and the same happened for red selection.
- Tested that tiles only appear in the grid.

Display Test:
- Tested that gameboard and tiles appeared in the intended location by mapping gameboard and tiles on graph paper with coordinates.
- Tested that the proper tiles are highlighted with selected.
- Tested that errors are highlighted with a red circle.
- Tested valid and invalid cases to ensure that the proper message was being displayed.
  - Clicked everywhere on the game window to ensure actions only occured wh
- Played many games, ending in all valid states - red wins, blue wins, tie.
- Saved and loaded while the game was in various states to ensure behaviour w

Testing Procedures:
- Play tested the game by clicking everywhere on the screen and ensuring only intended actions occur, alternated between selecting red and blue tiles,and created both valid and invalid game states.

10

CLASS VIEW



We decided to split our connect four game into 3 main parts: Model, View and Controller. By the definition of MVC, the user modifies the Controller, the Controller updates the Model, the Model tells the View what to render, and the View renders the graphics for the user to see.

This is exactly how we went about creating our game: the user modifies the MouseInput clsss, the MouseInput class modifies the GameBoard class, GameBoard class tells the View class what to render, and the View class uses lwjgl (light weight java game library) to render the graphics that the user will see.

The reason we decided to use MVC was because when making a game, it provides a great model for our **Separation of Concerns**. This makes it so we don't tend towards a **big ball of mud**!