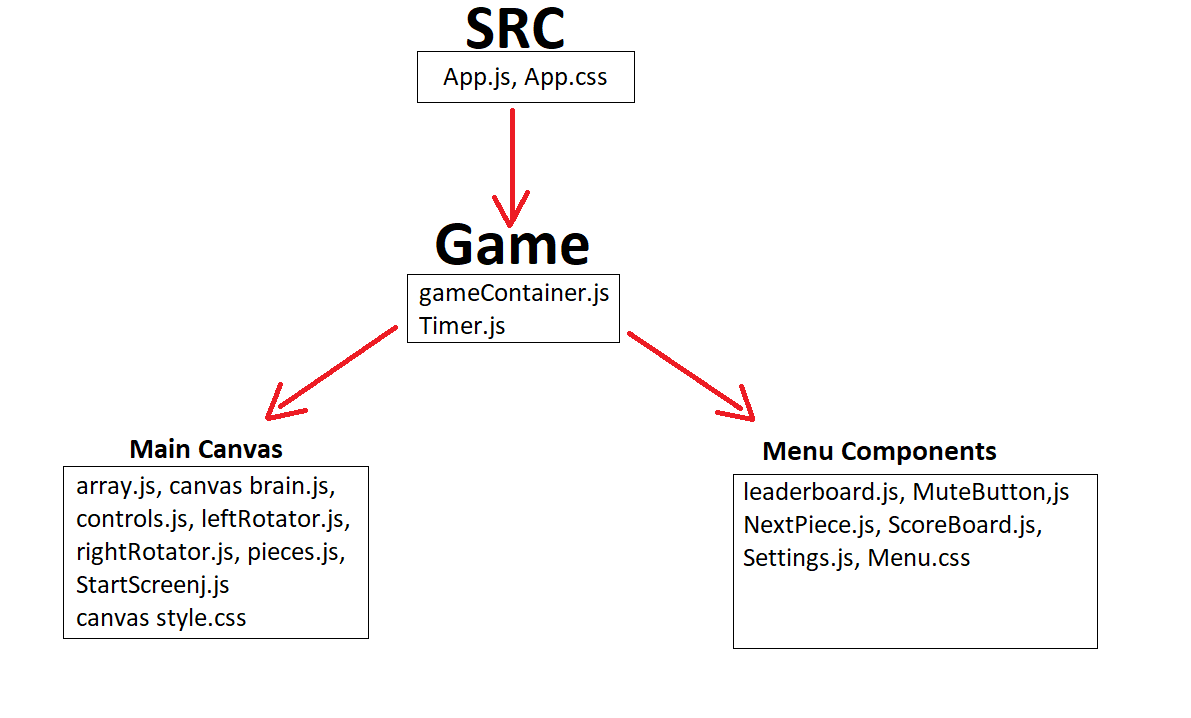
Documentation for tetris project.

1. File and folder structure:



1. SRC Contents:
2. App. – holds the most important states such as gameOn (if the game has started or not), states for settings such as language and volume (in progress)

App contains the following states and functions:

* 1. [gameOn] – tells components if a game has already been started or not. Serves also as a way to pause the game, and pause the Timer from counting time until next forced downward movement of current tetris piece.
  2. [isMuted] – if the game is muted or not. Will be used by the MuteButton component.
  3. [isSettingsOpen] – tracks if the settings are up or not. Will be used by the Settings component.

1. App.css – contains the style for the basic layout of the App. All css files may eventually be moved to App.css. Currently are split between 3 css files.
2. Game contents:
3. GameContainer – an intermediary component between the App and CanvasBoard component (canvas brain.js).

GameContainer controls the level of the game (tetris has levels.). The higher the level, the faster the piece falls and more points are earned for clearing rows.

Aside from that it holds some states for other components, and refs for itself.

GameContainer holds the following states and variables:

* 1. firstRenderRef - ref used to track the first render, so that sideEffects don’t run on first render. Present in many components and files.
  2. [currentLevel] – tracks the current Level of the game. Tetris has levels. Higher level means higher speed of the game (piece falls down faster) and more points from cleaned rows.
  3. [nextPieceState] – holds an object that has data for the next piece. Is used to display what the next piece is gonna be in the bottom right. Used by the NextPiece component (NextPiece.js).
  4. [linesCleared] – number of lines cleared by the last killrow() of CanvasBoard(canvas brain.js).

The number of rows cleared by the last move is important to count the score and increase level. Used directly by GameContainer and ScoreBoard component (ScoreBoard.js)

This state is set by CanvasBoard (canvas brain.js) whenever player cleans rows in the game.

* 1. rowsTo10 – Ref used to track how many rows have been cleared so far. Upon reaching 10 (or more) tetris level is increased by 1, and 10 is substracted form rowsTo10.current. Upon reaching another 10 the same operation is performed.
  2. [sendMovement] – state used to tell CanvasBoard (canvas brain.js) to move down.

This state is set from Timer component and only passed to CanvasBoard (canvas brain.js).

* 1. changeLevel – function used to increase the game Level by 1 upon reaching 10 rows cleaned.

GameContainer contains a single SideEffect:

* 1. This side effect reruns every time rows are cleared in the game. Checks if enough rows have been cleared to increase level. If the requirement has been met, level increases by one. \

1. Timer

Very important component that controls when the current tetris piece is moved downwards automatically.

How that works:

Gravity state tells how much time is needed for the next downward movement. Default state is 800 milliseconds.

[timeRightNow] state – Default is 0. It will be increased by 10 until It reaches Gravity.\*\*\*

When timeRightNow === Gravity:

setSendMovement changes [SendMovement] state – CanvasBoard moves the piece down.

That’s how the piece is moved downward automatically.

This action is performed in a sideEffect – if the requirement is not met (timeRightNow !== Gravity) – timeRightNow is increased by 10 after a 10 milliseconds delay, Timer rerenders and checks again.

This means that 80 rerenders occur between every activation of this mechanism.

Level increases the speed at which a tetris piece falls, which means it decreases the Gravity value.

This is performed by another sideEffect that reruns every time currentLevel changes.

The formula makes Gravity decrease rather rapidly so I may make a different formula to make the game more casual and average game length longer.

This sideEffect also sets timeRightNow to 0 so that timeRightNow will not exceed Gravity. If that happened, the piece would never move downwards and Timer would rerender indefinitely, 100 times per second.

1. Canvas brain and its associates.

Canvas brain is the most important component of the game.

It generates the board, spawns pieces, has event listener for movement and pause.

As many variables and functions have been exported to other files to keep the code as concise as possible.

Still ended up at some 500 lines.

I will attempt to explain how everything works in a function list.

1. Generating the board.
2. Principle for mutating the board
3. Countdown
4. How the app sees a tetris piece.
5. Spawning piece
6. Movement downwards
7. Movement sideways.
8. Rotation
9. Ghost piece
10. Instantaneous movement to the bottom.
11. Sending score

So let us begin.

1. Generting the board.

CanvasBoard uses StartArray to generate the board. It is contained within array.js and is declared there as a ready made object as opposed to one created by a function – that’s because I didn’t know back than about deep cloning arrays and had problems with mutating it later on.

StartArray is a 2D array.

It has 23 elements (from 0-22) to generate 23 (from top 0 to bottom 22) rows of the board.

Every one of the 23 “rows” has 10 elements inside (1 for each cell, 10 cells).

Every “cell” has a value: 0 (key value pair) and motion:false (key value pair).

Value will be used for displaying pieces. Motion is a relic from early development – moving piece was supposed to change this value to true and then false, however currently it is only used by the GhostPiece (point 9).

CanvasBoard maps through Start array to create 23 CanvasRow components. First 3 of them are invisible (CSS display:none). Invisible rows are needed for certain things later on.

CanvasBoard passes the corresponding array of the 2D StartArray to CanvasRow.

CanvasRow maps StartArray snippet from CanvasBoard to generate 10 CanvasCell components.

CanvasCell is the final component of CanvasBoard. The board has total of 230 cells.

1. Mutating the board.

Canvas board contains a Deep clone of StartArray. StartArray itself is Never change. It is never mutated. This deep clone is [GameArray] stated variable.

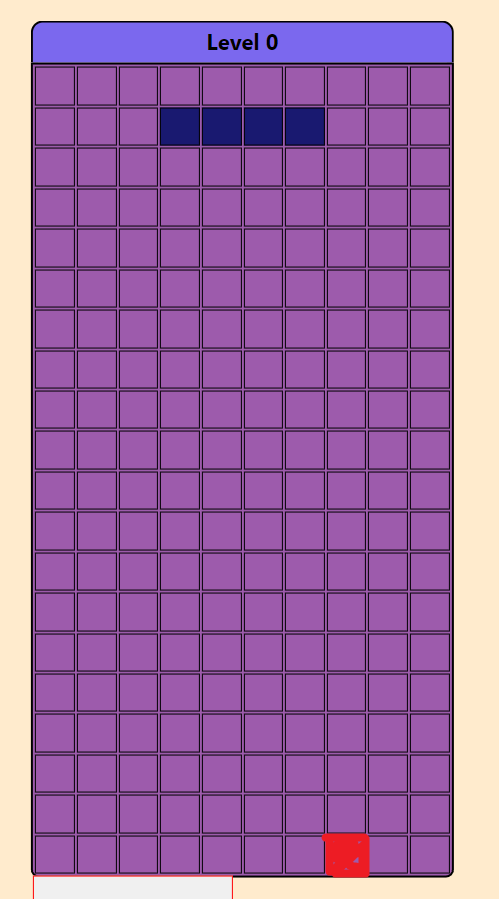
GameArray is passed from CanvasBoard to CanvasRows. CanvasRow does not use this component in any way.

CanvasRow passes GameArray to every one of its CanvasCells.

CanvasCell has a switch statement that control the className of the cell in question.

Every CanvasCell is being passed a single one, corresponding element of GameArray array.

Simple demonstration:



The red cell is in the last row, 3rd cell from the right.

That means it’s Y coordinate is 22 (I count top as 0, bottom as 22) and it’s X coordinate is 7 (0 to 9 from left to right).

So it will be passed GameArray[22][7].

Based on the value parameter (GameArray[22][7].value) switch statement will generate className for the cell.

Class determines the color of the cell.

An empty cell has a corresponding value of 0, and a color of no color (background of CanvasBoard is purple).

Let’s take the deep blue flat piece above the red square:

The flat blue piece above the red cell is composed of 4 cells, each with .value of 7.

GameArray is a mutable array. The way pieces are displayed, moved and rotated is basicly changing the corresponding YX coordinates in GameArray and then rerendering the component with className that will be adjusted accordingly to the changes.

So if I wanted to generate a yellow square piece, at 1st and 2nd row, I would change the GameArray[3][3].value = 1, GameArray[3][4].value = 1, GameArray[4][3].value = 1, GameArary[4][4].value = 1. (1st row is Y 3, 2nd row is Y 4).

The rest of CanvasBoard component is basicly mutating the GameArray and rerendering CanvasCells with changed values.

1. Countdown.

Upon clicking the startButton [BoardStatus] state is changed to false which:

Hides the start button

Triggers a sideEffect which sets [countdown] state to 3

[countdown] is displayed at the board at all times. However by default its value is “” so it’s not displayed.

Having set the value to 3, 3 is displayed

This triggers another sideEffect which triggers finalCountdown() function.

Final countdown reduces the countdown by 1 after a 1 second delay.

After the countdown reaches 0, finalCountdown runs for the final time.

Now it changes the value back to “”

SideEffect rerenders for the final time.

Seeing “” as the current value, countdown triggers trueGame() which spawns the piece and starts the game.