ECS GAMES

TETRECS

Created at Developed by

You!

Introducing TetrECS

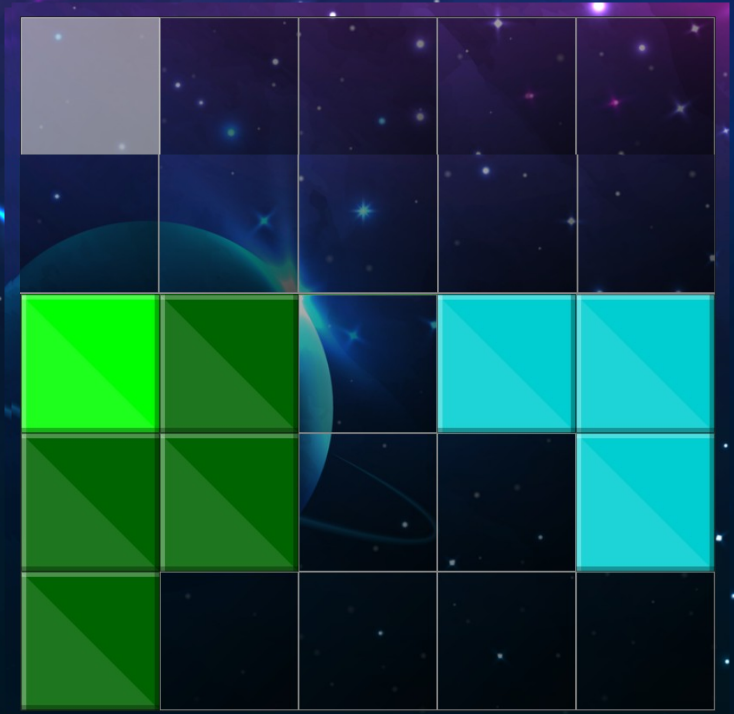
- We are going to put your new skills to the test by building a game

- Builds on everything you will be learning over the next few weeks
  - JavaFX
  - Custom Components
  - Graphics and Animation
  - Listeners, Properties and Binding
  - Communications
  - Media
  - Files

- In the labs, we will be building a similar application
  - You then apply the same concepts from the labs to this coursework

What is TetrECS?

- A fast-paced block placement game
- You have a 5x5 grid
- You must place pieces in that grid
- You score by clearing lines, horizontally or vertically



What is TetrECS?

- You can rotate pieces
- You can store a single piece to come back to later
- The more lines you clear in one go, the more points you get
- Every piece that you play that clears at least one line increases your score multiplier
- As your score goes up, so does your level, and you get less time to think
- If you fail to place a block, you lose a life
- Lose 3 lives, and you’re out of the game

Let’s see it in action!

- Can you work out what is going on at each stage?

You can do this!

- Nothing you just saw is beyond what you can do!
- The taught labs will prepare you for every concept needed for this coursework
  - But do make sure you attend them!
- This may seem like an ambitious coursework, but **you can do it!**
  - Everyone last year managed it, even those that didn’t think they could!
- Have confidence in yourself – you are all amazing and we believe in you!
- We are here to help you the best we can
- We will be running weekly tutorials to help with the coursework for those who want them

Coursework Sections

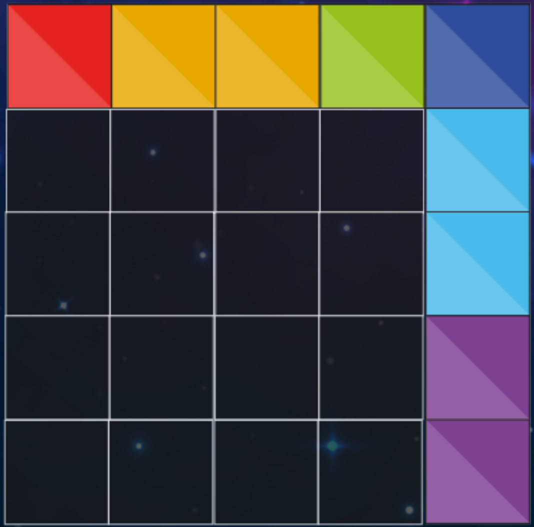
- The Coursework is split into sections, gradually increasing in difficulty

- Creating the Game Logic
- Building the User Interface
- Adding Events
- Adding Graphics
- Adding a Game Loop
- Adding Scores
- Adding an Online Scoreboard
- Adding Multiplayer

Getting Started

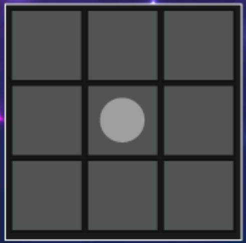
- Just like the labs
  - We will provide you with a basic skeleton application to get you going
  - We will provide you with the demo so you know what you’re trying to do
  - We will be helping you as much as we can on Discord
- Play with the demo and learn what it does, how it does it and how it works
- The coursework specification will guide you through the process
  - If you get stuck, revisit the labs and lectures covering that material

Let’s have a play



Create complete rows or columns to clear them. The more cleared at the same time, the more points you earn! The bonus will multiply as your clear more in a row!

You can use your mouse to click to place the tiles or use the cursor keys/WASD.

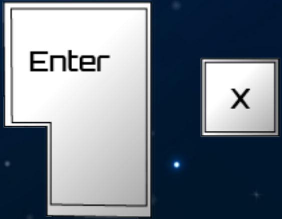


This shows the current piece. The circle represents what part of the piece will be placed. Click on it to ROTATE it

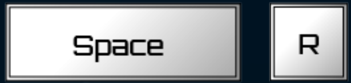
This shows the next piece. Click it to SWAP it with the current piece

You can rotate left and right with Q and E or Z and C or [and]

To go back to a previous screen, press Escape



Hit Enter, or X to DROP a piece



Hit Space or R to SWAP the upcoming tiles

Let’s have a play

- Connect to the VPN
- **git clone** http://ofb-labs.soton.ac.uk:3000/COMP1206/coursework-demo.git
- Run the tetrecs.jar java file
    - (Or tetrecs-legacy.jar if you’re on an Intel Mac)
- Give the single player challenge a go
- Note: The server is a little fragile, so please be nice to it

Now it’s your turn… Getting started!

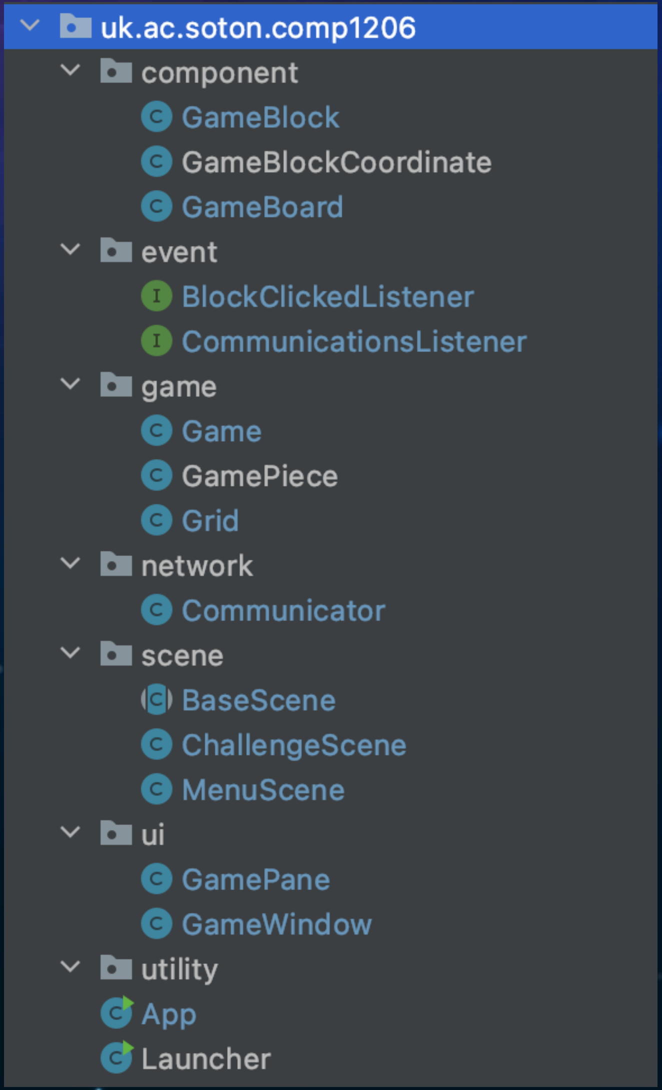
- **git clone** http://ofb-labs.soton.ac.uk:3000/COMP1206/coursework.git

- Has everything you need to get started
- Let’s take a look

- **Do not panic:** Most of this will be unfamiliar to you for now – but it’ll all make sense as we work through the labs, starting next Tuesday!

Introducing the Skeleton

- Components
- Events
- Game
- Network
- Scene
- UI
- App/Launcher

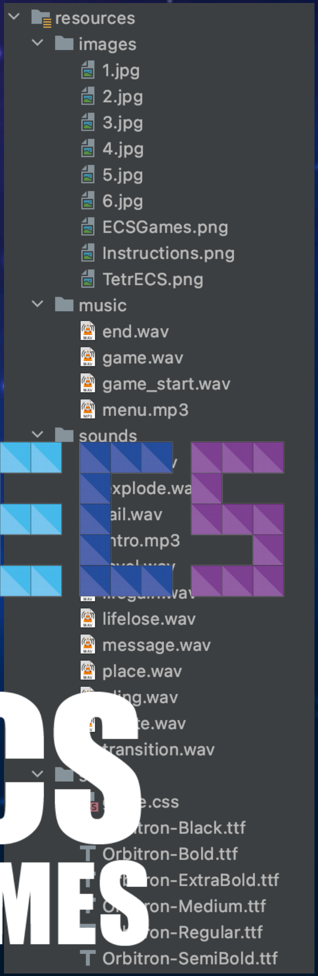


```
uk.ac.soton.comp1206
    component
        GameBlock
        GameBlockCoordinate
        GameBoard
    event
        BlockClickedListener
        CommunicationsListener
    game
        Game
        GamePiece
        Grid
    network
        Communicator
    scene
        BaseScene
        ChallengeScene
        MenuScene
    ui
        GamePane
        GameWindow
    utility
    App
    Launcher
```

All of the Assets!

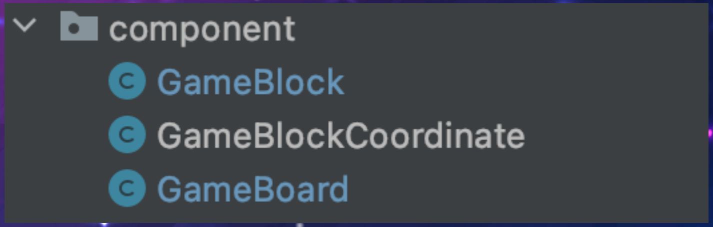
- You are welcome and encouraged to make the game look like how you want – with your own sound effects, graphics, music, theme

- However, for those who want an easy life, we are providing you with all the assets as found in the demo game!
  - The font
  - The graphics
  - The sounds and music
  - The CSS

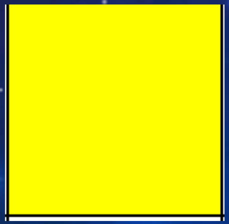


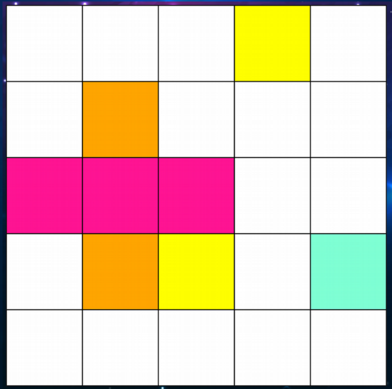
```
resources
  images
    1.jpg
    2.jpg
    3.jpg
    4.jpg
    5.jpg
    6.jpg
    ECSGames.png
    Instructions.png
    TetrECS.png
  music
    end.wav
    game.wav
    game_start.wav
    menu.mp3
  sounds
    xplode.wa
    ail.wav
    ntro.mp3
    lifelose.wav
    message.wav
    place.wav
    ing.wav
    e.wav
    ransition.wav
    e.css
    tron-Black.ttf
    Orbitron-Bold.ttf
    on-ExtraBold.ttf
    on-Medium.ttf
    on-Regular.ttf
    Orbitron-SemiBold.ttf
```

Components


component
GameBlock
GameBlockCoordinate
GameBoard

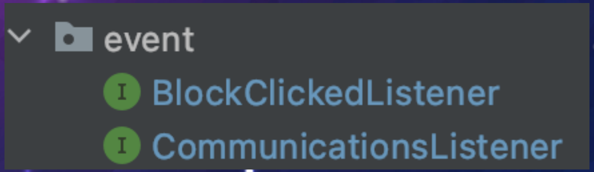
- **GameBlock**
  - Is a JavaFX custom component extending a Canvas
  - Displays an individual block
- **GameBoard**
  - Is a JavaFX custom component extending a GridPane
  - Holds all the GameBlocks in a grid
- **GameBlockCoordinate**
  - Holds an x and y column and row number of a GameBlock in the GameBoard





(3,0)

Events

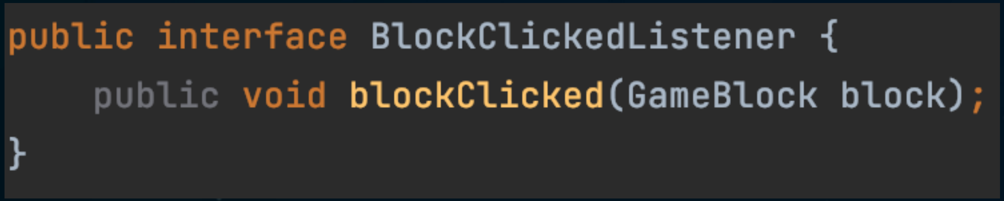


event
BlockClickedListener
CommunicationsListener

- We supply a couple of Listeners to get you started
- You will need to add more later

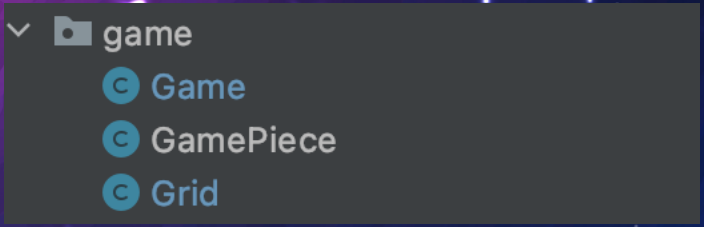
- **BlockClickedListener**: Handle a block being clicked
- **CommunicationsListener**: Handle receiving a message from the server

- Remember, these are just interfaces with a single method



```
public interface BlockClickedListener {
    public void blockClicked(GameBlock block);
}
```

Game

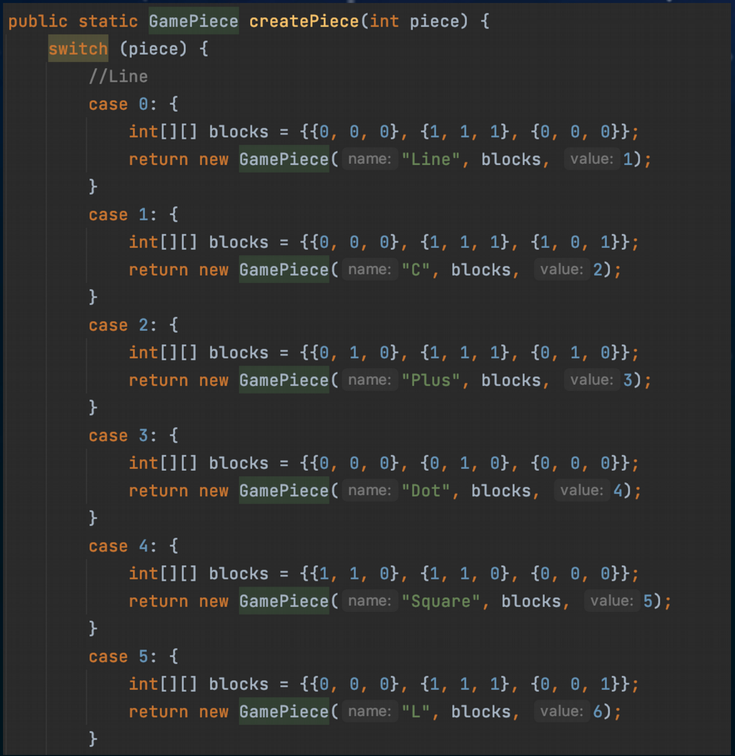

game
Game
GamePiece
Grid

- Holds the model and game logic

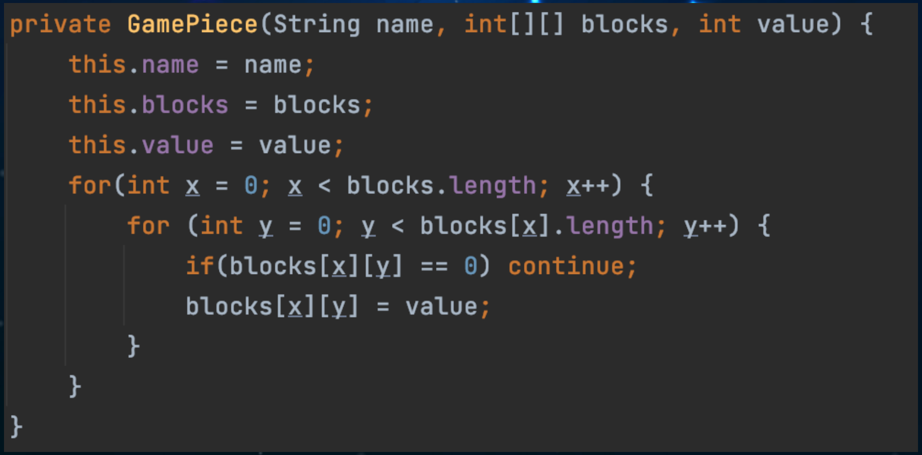
- **Game**: Handles the game logic
- **GamePiece**: Handles the model of a piece
- **Grid**: Handles the model of the grid
  - **This is displayed by the GameBoard**

Game: GamePiece

- Already has the 15 different pieces defined for you
- Every piece holds a 2D block array representing the piece
- Call the static **createPiece** method



```java
public static GamePiece createPiece(int piece) {
    switch (piece) {
        //Line
        case 0: {
            int[][] blocks = {{0, 0, 0}, {1, 1, 1}, {0, 0, 0}};
            return new GamePiece( name: "Line", blocks,  value: 1);
        }
        case 1: {
            int[][] blocks = {{0, 0, 0}, {1, 1, 1}, {1, 0, 1}};
            return new GamePiece( name: "C", blocks,  value: 2);
        }
        case 2: {
            int[][] blocks = {{0, 1, 0}, {1, 1, 1}, {0, 1, 0}};
            return new GamePiece( name: "Plus", blocks,  value: 3);
        }
        case 3: {
            int[][] blocks = {{0, 0, 0}, {0, 1, 0}, {0, 0, 0}};
            return new GamePiece( name: "Dot", blocks,  value: 4);
        }
        case 4: {
            int[][] blocks = {{1, 1, 0}, {1, 1, 0}, {0, 0, 0}};
            return new GamePiece( name: "Square", blocks,  value: 5);
        }
        case 5: {
            int[][] blocks = {{0, 0, 0}, {1, 1, 1}, {0, 0, 1}};
            return new GamePiece( name: "L", blocks,  value: 6);
        }
```

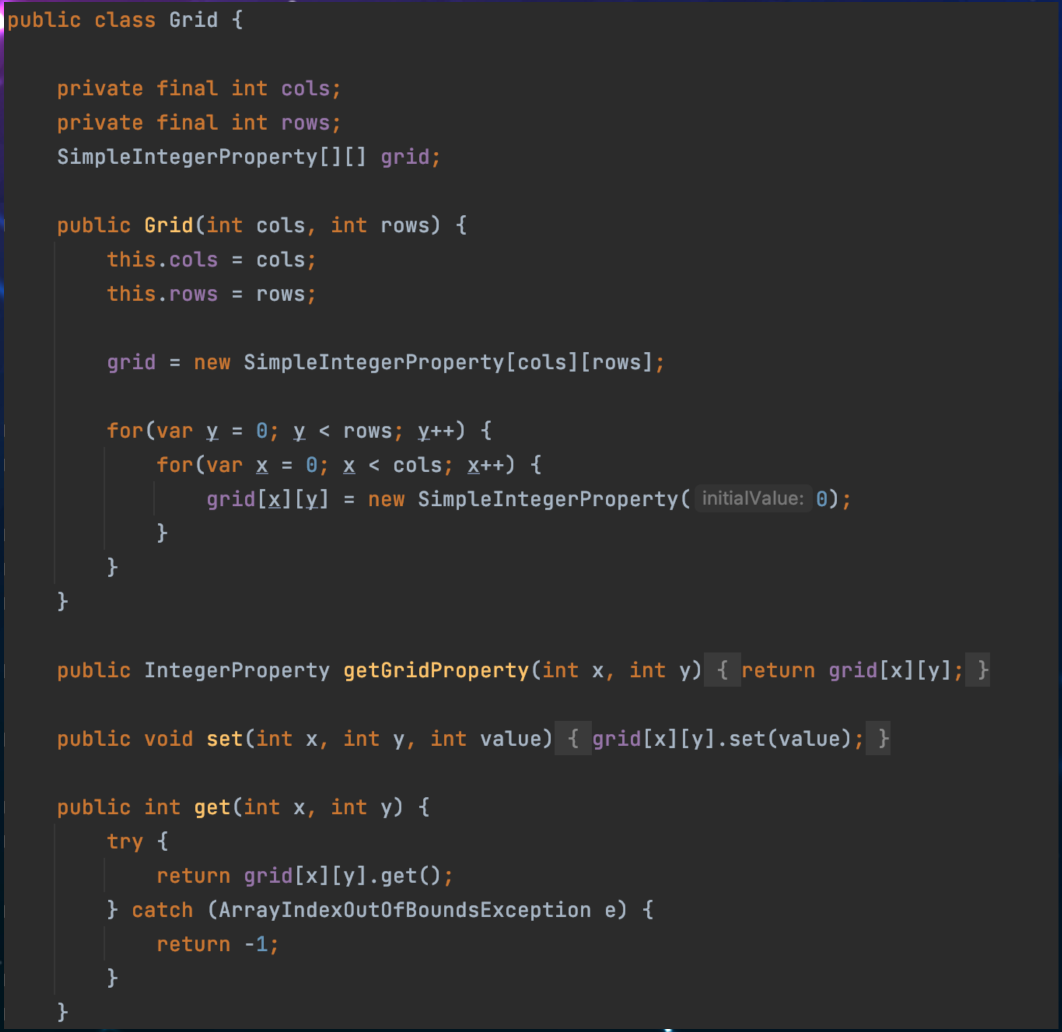


```java
private GamePiece(String name, int[][] blocks, int value) {
    this.name = name;
    this.blocks = blocks;
    this.value = value;
    for(int x = 0; x < blocks.length; x++) {
        for (int y = 0; y < blocks[x].length; y++) {
            if(blocks[x][y] == 0) continue;
            blocks[x][y] = value;
        }
    }
}
```

Game: Grid

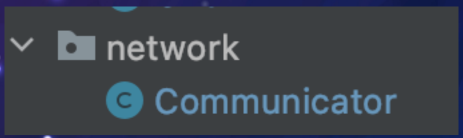
- Holds a 2D array of the grid
- SimpleIntegerProperties are used to represent each block
- The GameBoard binds to these properties

- Update the grid, update the graphical GameBoard representation



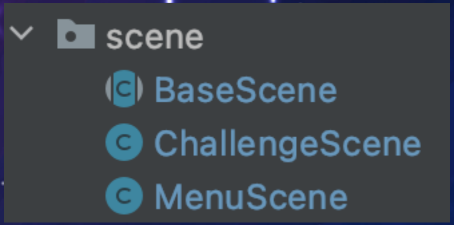
```java
public class Grid {

    private final int cols;
    private final int rows;
    SimpleIntegerProperty[][] grid;

    public Grid(int cols, int rows) {
        this.cols = cols;
        this.rows = rows;

        grid = new SimpleIntegerProperty[cols][rows];

        for(var y = 0; y < rows; y++) {
            for(var x = 0; x < cols; x++) {
                grid[x][y] = new SimpleIntegerProperty( initialValue: 0);
            }
        }
    }

    public IntegerProperty getGridProperty(int x, int y) { return grid[x][y]; }

    public void set(int x, int y, int value) { grid[x][y].set(value); }

    public int get(int x, int y) {
        try {
            return grid[x][y].get();
        } catch (ArrayIndexOutOfBoundsException e) {
            return -1;
        }
    }
```

Network



- Communicator
  - Works the same way as ECS Chat (to be introduced in labs)
  - You can add listeners to receive messages
  - You can send messages

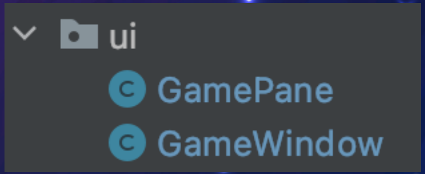
Scenes


scene
BaseScene
ChallengeScene
MenuScene

- Each Scene represents a “screen” in the game
  - You will be adding more later
  - For example
    - Intro
    - Instructions
    - Multiplayer (Lobby, Game)

- BaseScene
  - Provides a base scene the others inherit from
  - Basic functionality
- ChallengeScene
  - The single player challenge UI
- MenuScene
  - Displayed when the game is launched

UI

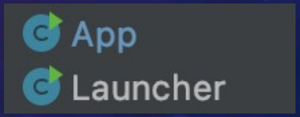

ui
GamePane
GameWindow

- Provides useful parts of the User Interface

- You shouldn’t really need to change these

- GamePane
  - This is a special pane which will scale all it’s internal content, adding padding to ensure correct aspect ratio
- GameWindow
  - The single window that switches scenes to change screen in the game

App / Launcher


App
Launcher

- App
  - The JavaFX Application
- Launcher
  - Starts the application

- You shouldn’t need to worry about these

So, what do you have to do?

- Creating the Game Logic
- Building the User Interface
- Adding Events
- Adding Graphics
- Adding a Game Loop
- Adding Scores
- Adding an Online Scoreboard
- Adding Multiplayer
- Extensions

Game Logic

- Add the logic to handle placing pieces
  - Can a piece be played?
  - Place a piece onto the grid
- Add the logic to keep track of pieces
  - Keep track of the current piece
  - Create new pieces on demand
- Add the logic to handle when a piece is played
  - Clear any lines

Game Logic (Details)

- **In the Grid class:**
  - Add a **canPlayPiece** method
    - With a given ***x*** and ***y*** of the grid will return true or false if that piece can be played
  - Add a **playPiece** method which
    - With a given ***x*** and ***y*** of the grid will place that piece in the grid
- **In the Game class:**
  - Add a **spawnPiece** method
    - Create a new random ***GamePiece*** by calling ***GamePiece*.createPiece**
  - Add a **currentPiece *GamePiece*** field to the ***Game*** class
    - This will keep track of the current piece
  - When the game is initialised, spawn a new **GamePiece** and set it as the ***currentPiece***
  - Add a **nextPiece** method
    - Replace the current piece with a new piece
  - Update the **blockClicked** method to play the current piece if possible, then fetch the next piece
  - Add an **afterPiece** method
    - To be called after playing a piece
    - This should clear any full vertical/horizontal lines that have been made

Build the User Interface

- Keep track of
  - Score
  - Level
  - Lives
  - Multiplier
- Show these in the UI
- Update them appropriately when events happen
  - Implement Scoring
  - Implement Multiplier
- Add Background Music

Build the User Interface (Details)

- Add **bindable properties** for the **score**, **level**, **lives** and **multiplier** to the ***Game*** class, with appropriate accessor methods.
  - These should default to 0 score, level 1, 3 lives and 1 x multiplier respectively.
- Add UI elements to show the score, level, multiplier and lives in the ***ChallengeScene*** by binding to the game properties.
- In the ***Game*** class, add a **score** method which takes the number of lines and number of blocks and call it in afterPiece. It should add a score based on the following formula:
  - score of lines * individual grid blocks * 10 * the multiplier
- Implement the multiplier
  - The multiplier is increased by 1 if the next piece also clears lines. It is increased **after** the score for the cleared set of lines is applied
  - The multiplier is reset to 1 when a piece is placed that doesn't clear any lines
  - If you clear 4 lines in one go, the multiplier increases once (now at 2x). If then clear 1 line with the next piece, the multiplier increases again (now at 3x). If you then clear 2 lines with the next piece, it increases again (now at 4x). The next piece you play clears no lines (multiplier resets to 1x)
- Implement the level
  - The level should increase per 1000 points (3000 points would be level 3)
- Create a ***Multimedia*** class
  - Add two MediaPlayer fields to handle an audio player and music player
  - Add a method to play an audio file
  - Add a method to play background music
  - Implement background music on the Menu and in the Game

Enhance The User Interface

- Make a better Menu
- Add an Instructions Screen
- Make a custom component to show a specific piece
- Add a listener for handling a next piece being ready
- Use the component the upcoming piece in the UI

Enhance the User Interface (Details)

- Create a **PieceBoard** as a new component which extends ***GameBoard***
  - This can be used to display an upcoming piece in a 3x3 grid
  - It should have a method for setting a piece to display
  - Add it to the Challenge scene
- Update the **MenuScene**
  - Add pictures, animations, styles and a proper menu
  - Add appropriate events by calling the methods on ***GameWindow*** to change scene
- Create a new **InstructionsScene**
  - This should show the game instructions
  - Add it to the menu
- In the **InstructionsScene**, add a dynamically generated display of all 15 pieces in the game
  - You can create a GridPane of **PieceBoard**s
- Add keyboard listeners to allow the user to press escape to exit the challenge or instructions or the game itself
- Create your own **NextPieceListener** interface which a **nextPiece** method which takes the next ***GamePiece*** as a parameter
- Add a **NextPieceListener** field and a **setNextPieceListener** method to ***Game***. Ensure the listener is called when the next piece is generated.
- Create a **NextPieceListener** in the ***ChallengeScene*** to listen to new pieces inside game and call an appropriate method.
  - In this method, pass the new piece to the **PieceBoard** so it displays.

Add Events

- Add the next tile in advance
- Add piece rotation
- Add piece swapping
- Add sound effects

Add Events (Details)

- Add a **rotateCurrentPiece** method in ***Game*** to rotate the next piece, using **GamePiece's** provided rotate method
- Add a **followingPiece** to ***Game***. Initialise it at game start.
- Update **nextPiece** to move the following peice to the current piece, and then replace the following piece.
- Add another, smaller ***PieceBoard*** to show the following peice to the ***ChallengeScene***
- Update the ***NextPieceListener*** to pass the following piece as well, and use this to update the following piece board.
- Add a **swapCurrentPiece** method to swap the current and following pieces
- Add a **RightClicked** listener and corresponding setOnRightClicked method to the ***GameBoard***
- Implement it so that right clicking on the main **GameBoard** or left clicking on the current piece board rotates the next piece
- Add sounds on events, such as placing pieces, rotating pieces, swapping pieces.
- Add keyboard support to the game, allowing positioning and dropping pieces via the keyboard

Add Graphics

- Add tiles to the game, not just squares
- Add hovering
- Add animations on clearing to show tiles cleared

Add Graphics (Details)

- Update the ***GameBlock*** drawing to produce prettier filled tiles and empty tiles
- Update the ***PieceBoard*** and ***GameBlock*** to show a circle on the middle square
  - Ensure that any pieces placed on the board are placed relative to this.
- Add events and drawing code to update the ***GameBoard*** and ***GameBlock*** to highlight the block currently *hovered* over
- Create a new **fadeOut** method on the ***GameBlock***
  - Using an AnimationTimer, use this to flash and then fades out to indicate a cleared block
- Create a new **fadeOut** method on the ***GameBoard*** which takes a ***Set*** of ***GameBlockCoordinates*** and triggers the **fadeOut** method for each block
- Create a **LineClearedListener** which takes a ***Set*** of ***GameBlockCoordinates*** (that hold an x and y in the grid of blocks cleared) and add it to the **Game** class to trigger when lines are cleared.
- Use the **LineClearedListener** in the ***ChallengeScene*** to receive blocks cleared from the ***Game*** and pass them to fade out to the ***GameBoard***

Add a Game Loop

- Add a timer to count down how long there is until the piece must be placed
- When the timer runs out, move on to the next piece and lose a life
- Show the timer in the game UI

Add a Game Loop (Details)

- Add a **getTimerDelay** function in ***Game***
  - Calculate the delay at the maximum of either 2500 milliseconds or 12000 - 500 * the current level
- Implement a ***Timer*** or ***ExecutorService*** inside the ***Game*** class which calls a **gameLoop** method
  - This should be started when the game starts and repeat at the interval specified by the **getTimerDelay** function
  - When **gameLoop** fires (the timer reaches 0): lose a life, the current piece is discarded and the timer restarts.
  - The timer should be <u>**reset**</u> when a piece is played, to the new timer delay (which may have changed)
- Create a **GameLoopListener**
  - and a setOnGameLoop method to link it to a listener
  - Use the GameLoopListener to link the timer inside the game with the UI timer
- Create and add an animated timer bar to the ***ChallengeScene***.
  - Use Transitions or an AnimationTimer to implement the timer bar
  - The ChallengeScene should use the GameLoopListener to listen on the GameLoop starting and reset the bar and animation
  - The timer bar should change colour to indicate urgency.
- When the number of lives goes below 0, the game should end

Add Scores

- Create a new Scores Screen
- Save and read high scores to a scores file
- Prompt for a new on getting a high score

Add Scores (Details)

- Create a new ***ScoresScene***
  - Add a **localScores** ***SimpleListProperty*** to hold the current list of scores in the Scene
  - Use **FXCollections.*observableArrayList*** *to make an observable list*
  - Use **Pair**<String,Integer> to represent a score
  - Add the relevant method to start it into GameWindow
  - Switch to it when the game ends
  - You should pass through the Game object, containing the final game state
- Create a new **ScoresList** component and add it to the ***ScoresScene***
  - Use a ***SimpleListProperty*** inside the ScoresList
  - Update the scores when the list is updated
  - Bind the ScoresList scores to the ScoresScene scores list
  - Add a **reveal** method which animates the display of the scores
- Write a **loadScores** method to load a set of high scores from a file and populate an ordered list
  - A simple format of newline separated name:score will suffice
  - Update the ScoresScene score list with the loaded scores (which will update the ScoresList)
- Write a **writeScores** method to write an ordered list of scores into a file, using the same format as above.
- If the scores file does not exist, write a default list of scores to the file.
- When the ***ScoresScene*** starts
  - Load the ordered list list of scores and link the scores to the ScoreList
  - If the score contained inside the ***Game*** beats any of the scores, prompt the user for their name, insert it into the list at the correct position, display the scores and update the saved file.
  - Reveal the high scores
- Add a **getHighScore** method to the ***ChallengeScene***
  - Get the top high score when starting a game in the ChallengeScene and display it in the UI
  - If the user exceeds it, increase the high score with the users high score.

Add Online Scoreboard

- Receive online scores from the server
- Include these in the Score Screen
- If the new score beats the current scores, submit it to the server

Add Online Scoreboard (Details)

- Add a **remoteScores** SimpleListProperty in the Scene
- Add **loadOnlineScores** method to ScoresScene
  - Use the ***Communicator*** to request HISCORES when entering the ScoresScene
- Use a ***CommunicationsListener*** within the ***ScoresScene*** to parse the high scores and create a second ordered list of online high scores.
  - Update the **remoteScores** list
  - You will need to ensure you only trigger displaying the high scores and checking the high scores occurs after the scores list arrives!
- Add a **writeOnlineScore** method
  - Use the ***Communicator*** to submit a new HISCORE if the user has beaten the previous high scores, as well as inserting their score into the list.
- Add another ***ScoresList*** component bound to the remoteScores property to display the online scores list alongside the local high scores list.

Add Multiplayer

- Implement the Lobby System
  - Find all games
  - Create a new Game
  - Join a game that exists already
  - Chat
- Implement the Multiplayer Gameplay
  - Create a leaderboard against the people you’re playing with
  - Send and receive blocks from the server
  - Send and receive game updates

Add Multiplayer: Lobby

- Create a Multiplayer menu option and create a corresponding **LobbyScene**.
  - The LobbyScene on opening should start a repeating timer requesting current channels using the Communicator from the server.
  - Add a listener to the Communicator to handle incoming messages
    - You will need to handle the different commands that are receiveved
  - When channels are received, the LobbyScene UI should be updated to show available channels.
- Add a button to start a new channel,
  - Prompt the user for a channel name
  - Submit it using the Communicator
- Allow the user to JOIN a channel by sending the corresponding command to the Communicator.
  - Update the UI to show the channel and the users currently inside that channel.
- Add a chat box to allow sending and receiving chat messages with people in that channel.
- Add buttons to start the game (if the user is the host) and leave the channel
  - Send appropriate messages when these buttons are clicked to the server
- Add error handling on receiving ERROR messages from the server.

Add Multiplayer: Gameplay

- Create a custom **Leaderboard** component which extends the **ScoreList**
- Create a **MultiplayerScene** which extends the ***ChallengeScene***.
- Update the UI to include chat and the leaderboard and remove less important elements if needed.
- Create a **MultiplayerGame** which extends the ***Game***.
  - Using a queue or similar, request pieces as needed from the server and use these to populate the current and upcoming pieces.
- The **MultiplayerGame** should listen for incoming pieces, score updates and chat messages.
  - When score updates are received, update the list which should be bound to the leaderboard and update automatically.
- Send appropriate updates using the **Communicator** on board changes, score changes, live changes, getting a game over and leaving the game.
- Update the leaderboard to track number of lives and when people get a game over, crossing them off as they are eliminated.
- Update the **ScoresScene** to substitute showing local scores for the multiplayer game scores, held in the ***MultplayerGame*** object

Extensions

- Add further polish, animations and effects to the user interface
- Add statistics to the ScoreScene and save and retrieve them between games
- Allow customising the single player challenge with different options
- Add a settings menu to change volume, resolution and other settings. Store and retrieve this from a config file
- Add custom theme support to change the appearance, sounds and music
- Display other peoples boards in multi player
- Implement powerups in single player
- Implement a duel mode where you can play against a bot
- Implement your own compatible multiplayer server in Java
- Introduce your own new server-side multiplayer features
- **Anything else you think will impress us!**

- **There are up to 5 marks available for extensions – you can do a small number or even one extension well to get all 5 marks, or work through multiple extensions. Marks will be awarded based on the level of challenge or number of extensions and the technical achievement.**

Marking

- Worth 40% across 80 marks

- The Basics: 2
- Creating the Game Logic: 8
- Building the User Interface: 6
- Enhancing the User Interface: 8
- Adding Events: 6
- Adding Graphics: 7
- Adding a Game Loop: 6
- Adding Scores: 6
- Adding an Online Scoreboard: 4
- Adding Multiplayer: 12
- Extensions: 5

- In addition:

- Code Quality: 5 marks

- Understanding: 5 marks

Submission

- You will be required to submit a fully working **Maven project**
- It must be possible for us to compile and build your project

- You will be required to demonstrate your game in the lab session to a demonstrator, who will mark your functionality
  - They will be marked in the Lab slot on 25th April – make sure you are there in your 9am or 11am slot!
- You will also be required to submit a **5 minute video** walkthrough of your game showing it meeting all the criteria provided

- Submission is straight after Easter
- Deadline for demoing: 25/04/2023 - Tuesday
- Deadline for submission: 28/04/2023 - Friday

Other Notices

- Specification
  - You can find the full specification here: https://secure.ecs.soton.ac.uk/noteswiki/w/COMP1206/Coursework
  - The specification will be updated to clarify any FAQs so check frequently
- Support
  - Use **HELPDESK** – 1:1 support for your coursework!
    - On Discord
    - On helpdesk@ecs.soton.ac.uk
  - Use the #coursework channel for questions
    - We will continue to help you there!
    - Provide us much detail as you can
  - We will run a coursework tutorial session

Good luck!

- Any questions?