# Introduction

## Goals of project

Learn programming

Deliver a real application

## Story-based development

Based on agile development

Each story adds some new value to user

Each story kept to the minimum complexity that adds some value

## Learning approach

Programming patterns, tools & techniques introduced only as they are used

Where possible, each is introduced in a context outside the app

The student is then invited to apply this learning, with some hints and guidelines

Model code available at the end of each module

## Technical approach

Web-based

Microsoft tooling platform

TypeScript rather than JavaScript

## Professional perspective

What’s realistic and what’s not

Stories have been nicely ordered

While there is some change to code, this is reduced because we know where we are going.

Watch out for ‘Professional Perspective’

# Create a web page with a welcome screen

## Learnings: patterns & techniques

Use Visual Studio

Create a web-based project

The index page

The run-edit cycle

Some simple Html

## Exercises

Create new typescript project

Run it unmodified.

Explain parts, including: TypeScript vs. JavaScript

Empty App.ts for now

Modify Index

Modify CSS

Show that if app is still running, changes can be picked up by refreshing the browser

## Code written

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<title>My board game</title>

<link rel="stylesheet" href="app.css" type="text/css" />

<script src="app.js"></script>

</head>

<body>

<h1>Reversi Game</h1>

<p>Look up the <a href="http://www.yourturnmyturn.com/rules/reversi.php" target="\_blank">rules</a></p>

</body>

</html>

app.ts - empty

app.css – as is

# Draw the outline of a board

Large green square with an outline. Text to the side

## Learnings: patterns & techniques

Canvas

Script files

TypeScript (& JavaScript)

Renderer

Rectangle

Fill & stroke

Arc

Procedural programming - executed in order

Global variables (which we want to avoid)

## Exercises

Add code

Explain code

What happens if:

* You change numbers in the rectangle method
* You change the order of the statements
* Draw another rectangle of different size, colour & location
* You spell canvas wrong in various places.
* Step through code

## Code written

index.html

…

<body>

<canvas width="500" height="500"></canvas>

<h1>Reversi Game</h1>

…

app.ts

var canvas: HTMLCanvasElement;

var renderer: CanvasRenderingContext2D;

window.onload = function () {

canvas = document.getElementsByTagName("canvas")[0];

renderer = canvas.getContext("2d");

//Draw background

renderer.fillStyle = 'green';

renderer.fillRect(0, 0, 480, 480);

//Draw outline

renderer.strokeStyle = 'black';

renderer.strokeRect(0, 0, 480, 480);

}

app.css

canvas {

float: left;

}

# Create squares in specific positions

## Learnings: patterns & techniques

Separation of concerns - domain logic from presentation

Beginnings of object modelling

Classes and instances

Constructor

Writing a generalised function

Passing params

## Code written

app.ts

const squareSide: number = 60;

var canvas: HTMLCanvasElement;

var renderer: CanvasRenderingContext2D;

import Square = model.Square;

window.onload = function () {

canvas = document.getElementsByTagName("canvas")[0];

renderer = canvas.getContext("2d");

var sq = new Square(0, 0);

drawing.drawSquare(sq, renderer);

sq = new Square(0, 1);

drawing.drawSquare(sq, renderer);

var sq = new Square(1, 0);

drawing.drawSquare(sq, renderer);

}

drawing.ts

module drawing {

import Square = model.Square;

export function drawSquare(sq: Square, renderer: CanvasRenderingContext2D) {

//Draw background

renderer.fillStyle = 'green';

renderer.fillRect(sq.col \* squareSide, sq.row \* squareSide, squareSide, squareSide);

//Draw outline

renderer.strokeStyle = 'black';

renderer.strokeRect(sq.col \* squareSide, sq.row \* squareSide, squareSide, squareSide);

}

}

model.ts

module model {

export class Square {

constructor(public col: number, public row: number) {

}

}

}

# Create a whole board

## Learnings: patterns & techniques

Loops

Instance method (function on an object)

Using external libraries, installed via NuGet (lodash & definitely typed - but delete duplication)

## Code written

### Board object

export class Board {

constructor() {

this.squares = [];

for (var col = 0; col < 8; col++) {

for (var row = 0; row < 8; row++) {

this.squares.push(new Square(col, row));

}

}

}

private squares: Square[];

//If the coordinates lie outside of the board boundaries, returns 'undefined'

public getSquare(col: number, row: number): Square {

return \_.find(this.squares, sq => sq.col === col && sq.row == row);

}

}

### drawBoard function

export function drawBoard(board: Board, renderer: CanvasRenderingContext2D) {

for (var col: number = 0; col <= 7; col++) {

for (var row: number = 0; row <= 7; row++) {

const sq = board.getSquare(col, row);

drawSquare(sq, renderer);

}

}

}

# Set up the starting position

## Learnings: patterns & techniques

Enum

Importance of naming & renaming

## Code written

### new constant PieceRadius (in App)

const pieceRadius = 25;

### New enum (in models)

export enum Side { black, white }

### Function to get drawing colour from Side value

import Side = model.Side;

export function getColourForSide(side: Side): string {

return side === Side.black? 'black' : 'white';

}

### Modify square to know if occupied

export class Square {

constructor(public col: number, public row: number) {

this.occupiedBy = null;

}

public occupiedBy: Side;

}

### new drawPiece function

export function drawPiece(

sq: Square,

colour: string,

renderer: CanvasRenderingContext2D) {

var centreX: number = sq.col \* squareSide + squareSide / 2;

var centerY: number = sq.row \* squareSide + squareSide / 2;

renderer.fillStyle = colour;

renderer.beginPath();

renderer.arc(centreX, centerY, pieceRadius, 0, 2 \* Math.PI);

renderer.fill();

}

### Modify drawSquare to draw piece if occupied

export function drawSquare(sq: Square, renderer: CanvasRenderingContext2D) {

//Draw background

renderer.fillStyle = 'green';

renderer.fillRect(sq.col \* squareSide, sq.row \* squareSide, squareSide, squareSide);

//Draw outline

renderer.strokeStyle = 'black';

renderer.strokeRect(sq.col \* squareSide, sq.row \* squareSide, squareSide, squareSide);

if (sq.occupiedBy != null) {

var colour: string = getColourForSide(sq.occupiedBy);

drawPiece(sq, colour, renderer);

}

}

### Set up starting pieces in the Board’s constructor

this.getSquare(3, 3).occupiedBy = Side.white;

this.getSquare(4, 4).occupiedBy = Side.white;

this.getSquare(4, 3).occupiedBy = Side.black;

this.getSquare(3, 4).occupiedBy = Side.black;

# Move a cursor around the board

## Learnings: patterns & techniques

Reacting to events (we already did this with on-load but less consciously)

Using an external library (LoDash)

Using NuGet

Writing automated test

Switch Case statement

Functional programming

## Code written

Separate drawing, eventManagement and model files

Draw a cursor

Function to locate square within board

Unit test for that function

Add instructions into Index page

# Place a piece on the board

# Manage players and turns -

## Learnings: patterns & techniques

Refactoring?

Placing text into a web-page

## Code written

GameManager class

Drawing functions

Status section on Index

# Can only move onto an empty square

## Learnings: patterns & techniques

Rules and constraints

## Code written

Function for is valid move

Changing the colour of the cursor

# New piece must be adjacent to a piece of opposite colour

# Flip any captured pieces

# 12.Move only valid if pieces are captured

# 13. Keep track of piece counts

# 14. Allow move to be skipped

# 15. Identify the end of the game

# 16. Automated player

## Learnings: patterns & techniques

Sub-classing: automated player and manual player

## Notes

Get list of valid moves.

First, pick one at random

Next option, go for move with highest number of pieces gained, otherwise random

# TO THINK ABOUT

When to introduce the idea of a Player object

Game master need to ask Players for next move

Introduce idea of an automated player

AutomatedPlayer1

# 16. Match

Set up two players at the beginning

Specify number of games

Execute games, keeping cumulative score

Output results.

## Learnings: patterns & techniques

Executing model without a UI