Minutes

For Sussex Software Engineering 2020, Team 41

This is our combined process document and group report.

1. Meeting: Mon, Feb 10 at 16:00

Attendees

- Steven Liang, Computing for Digital Media
- Chris Mouzouris, Games and Multimedia Environments
- Jacob Poole, Computer Science
- Greg Pyemont, Computer Science
- David Pomerenke, Computer Science (keeping minutes)

Umar Shafii Ndayako (Computing for Digital Media) is missing .

Agenda

1. Introducing Ourselves
2. Discussing Tools
3. Creating a Project Plan
4. Scheduling Meetings
5. Tasks

1. Introducing Ourselves

→ Attendees.

2. Tools

Unity	Steven, Greg & Chris have used it
Java	everyone knows to use it, Jacob is strong in it
3D Modeling	Steven, Umar & Chris
Javascript / React	Jacob, Greg, David know a bit but no too much

Databases	everyone has attended the Databases module
Github	everyone knows it

We decide to use **Unity,** because it is suitable for creating an appealing user interface. It uses **Java** for Game Logics, which is convenient. It can be used with Github.
We also decide to use a database for the representation of all the game data, as these data are quite complex. It should use **SQL** because we know how to use that, but we do not yet settle for a specific database technology.
We are using Slack for chatting, Google Docs for keeping minutes, and want to use Github for code colaboration.

3. Creating a Project Plan

The project is **due on May 1st**, including a presentation of the running game, so the game should ideally be running a bit earlier. This leaves us almost 11 weeks. 2 of these weeks are holidays, and there may be strikes during other weeks.
We discuss and identify the key steps. They are in rough but not exact chronological order. For example, logics, database and UI may have to be developed in parallel.

- board
- logics
    - throw dice
    - actions on tiles
    - testing
- database
    - players
    - assets
    - bank
    - testing
- user interface
    - whose turn is it
    - options: buy, take card, etc.
- images for houses, players, etc.
- AI
- testing

We start to make a PERT chart out of this list:

Task	Duration	Early Start	Early Finish	Late Start	Late Finish	Critical?
Project Plan	1d	10.02.	10.02.	19.02.	19.02.	yes

Requirement Analysis	1d	19.02.	19.02.			yes
Board						
Logic						
Database						
User Interface						
Images						
AI						
Testing						

The next step is requirement analysis, which we have not yet covered in the lectures, but it will probably be covered in the seminars starting next week.

4. Scheduling Meetings

Some have a seminar on Mondays, some on Fridays, beginning the same week, so there is no way meeting during the week and being synchronized seminar-wise.
We decide to meet **each week** on **Wednesdays at 1pm** in the library. Ideally, the room should be booked for more than one hour. If we do good work during the week, we might nkeep the meetings very short.

5. Tasks

Until next week:

- We all get comfortable with Unity, specifically Jacob and David, who have not used it yet.
- Prepare an ER diagram, collaboratively in the Google Docs.
- Prepare PERT chart a bit.
- David writes down the minutes (= the meeting protocol).

2nd Meeting: Wed, Feb 19, 13:00

Attendees

Steven, Greg, Chris, Umar and David (keeping the minutes) are present. Jacob is sick and excused.

Agenda

- PERT chart
- Review of ER chart
- Discussion of seminars
- Re-Evaluation of Tools
- Planning of first sprint cycle

We decide that a PERT chart for the whole project is nothing we need to start our first sprint, and thus drop this agendum for now.

Discussion of Seminars

Greg, Chris and David have already had their seminar. It has turned out in the seminars that a database may not be a good idea for the given user requirements, so we decide to drop this idea and handle all the data with appropriate classes and data structures.

Re-Evaluation of Tools

It has turned out that Unity does not support Java, but only C#, which is known to be similar to Java but which nobody has used so far, and JavaScript, which only David is comfortable with. We assess three alternative strategy adaptations:

- Using unity with C#.
- ~~Using Unity with JavaScript.~~
- Using Java with JavaFX.

We decide that Unity is way better for graphics & UI than JavaFX, so we want to stick with it. We will thus use C#, possibly in combination with JavaScript, in Unity.
*(**Update** 22.02.: Unity seems to have dropped JavaScript support in 2017. David)*
Besides this issue, we have now created a GitHub organization and repository at https://github.com/teamfourtyone/property-tycoon. We also have a project Kanban in there.

Planning of First Sprint Cycle

We want to start our first sprint cycle today. As we have never done one before and the work will not be too much, one week will be a good duration.
We discuss whether it is necessary to write down our own version of the user requirements and finally agree that Mr Raffles' version of the user requirement is sufficient for our purposes. In order to create task cards, we adapt and extend the plan from item (3.) of our first meeting to specify the properties and methods of the classes which we will need to code.

Board class (Steven)

Static class.

- contains an array of Tile class instances of length 4 * 10 = 40

Game class (David)

Static class, for general game-play mechanisms / data.

- method: initialize()
    - shuffle card piles
- 2 card piles (stacks) → card ids
- method: move player(p)
    - dice roll
        - random number
        - check for double → repeat
        - store number of doubles in a row and check for jail
    - update position of the player p
    - check for type of tile where player p ends up and call according functions
        - call landing action of tile
        - visually update (see later)

Player class (?)

We will use 2-6 instances of it in a game.

- position of the game piece on the board
- balance
    - initial value: 1 500
- array of street properties → street property class ids
- nr of having passed go
- nr of blocked rounds because of jail (0 normally)

Tile class (Greg)

We will have 40 instances of it, stored in the Board class.

- tile id
- method prototype: landing action (different for each subclass)

Street / Property subclass

- street id is tile id
- owner → player id
- colour
- nr of houses
- mortgaged or not
- landing action: check owner
    - if free: ask whether buy (later)
        - if buys:
            - pay bank

            - change owner id of street
        - else: auction (we'll see later)
    - if owned
        - check nr of houses & colours and calculate price
        - add to owner balance, subtract from player balance
            - if player balance too low, needs to sell property
        - if owned by player: check for upgrade
            - if upgrade possible: ask for upgrade

Free parking subclass

- balance
- landing action:
    - add free parking balance to owner balance
    - set free parking balance 0

Draw card subclass

- landing action: call card.draw()

'Go' tile subclass

- landing action:
    - balance of player increases by 200
    - 'nr of having passed go' of player increases

Card class (Steven)

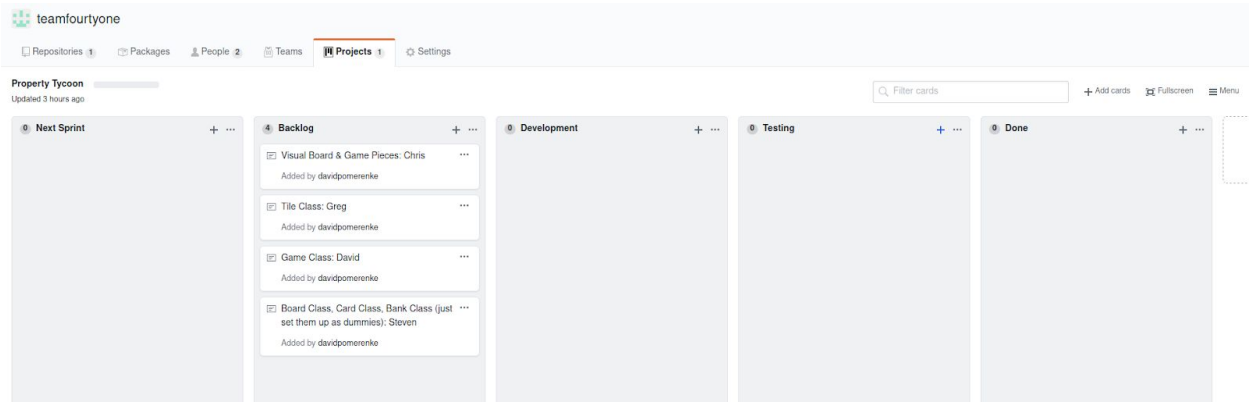
- card id
- method draw
    - call action of the card (implement later)
    - put the card to the bottom of the pile
- Pot Luck Card sublass
- Opportunity Knock Card subclass

Bank class (Steven)

Static class.

- balance:
    - initial value: 50 000
- (property)

One additional task is to create a visual representation of the board in Unity, which will be done by Chris.
We subsume Board, Card, and Game class under one task, and create tasks according to the classes in our Github project Kanban:



We are a bit insecure how we should separately work at different components depending on each other. We decide that everyone just codes / designs for themselves during the sprint, and we will integrate the results together next week.

3rd Meeting: Wed, Feb 26, 13:00

Attendees

Steven, David, Umar, Greg(1h late).

Sprint 1 Review

- Game and Player class prototypes are working (though very crude).
- Graphics?
- Other task cards have not been pursued.

System Requirements

Functional Requirements

Reference	Requirement	Mandatory / Required

F1	- 2-6 players
	- tokens:
	  - boot
	  - smartphone
	  - goblet
	  - hatstand
		mandatory

	- cat - spoon	

F2	- game play
	  - dice rolling
	  - two dices
	  - double → another turn
	  - three doubles → prison
	  - moving
	  - clockwise
	  - no option to retire, only bankruptcy
		mandatory

F3	- game initialisation
	  - all players on go
	  - 1500£ each
	  - shuffle “pot luck” and “opportunity knocks” cards
		mandatory

F4	- action cards
	  - “get out of jail free” card
	  - cards are stacks
	  - cards have instructions
		mandatory

F5	- board spaces
	  - properties
	  - go → +200£
	  - prison / just visiting
	  - pot luck
	  - opportunity knock
	  - free parking → player gets accumulated money
	  - specific instruction space
		mandatory

F6	- property
	  - initially at bank
	  - buying
	  - property can be bought after first go pass
	  - only on landing on the property
	  - price is transferred player → bank
	  - otherwise auction
	  - each player bids
	  - bank sells to highest bidder
	  - no bids → no sell
	  - bidding requires having passed go
	  - landing on property of other player → pay price
	  - owner owns whole colour group → double rent
	  - houses hotels → rent depends on imported property details
		mandatory

	- unable to pay → must sell
	  - for original value
	  - only when no houses
	  - houses can also be sold for original value
	 - unable to pay or sell → leave game, remove player token
	- can be improved by player at end of their move
	 - only if all properties of the colour are owned by player
	 - no house number difference > 1 in same colour group
	 - hotel counts as 5th house
	 - max per property is one hotel
	- can be mortgaged
	 - half price is returned
	 - no rent collection when mortgaged
	 - can be sold for half of original price	

F7	- fines accumulate on free parking space	mandatory
F8	- jail - costs 50£ to get out → goes to free parking fines, token moves to “just visiting”, normal turn next round - otherwise no move for 2 rounds, no rent collection from other players	mandatory
F9	- AI game player - 0 to all game players can be played by AIs, high-speed simulation is possible - at least randomly selects among all the actions which the human player can do	mandatory
F10	- AI player takes financially smart decisions	desirable
F11	- upload data from XXX file format. - documentation of the file structure for importing	mandatory
F12	- UI elements of all players - current worth - owned property assets	mandatory
F13	- test correct working of game	mandatory

Non-Functional Requirements

Reference	Requirement	Mandatory/ Required
NF1	Unity 2019.3 shall be used to create an intuitive GUI using c#	Mandatory
NF2	Game shall support windows/mac	Mandatory
NF3	Game may support other operating systems ie, linux, mobile	Desired
NF4	Game should run without errors	Mandatory
NF5	At initialisation, the game shall be able to quickly retrieve board and card data from external excel sheet and set the game up correctly	Mandatory
NF6	Players shall not be able to … *influence backend [word better]*	

Domain Requirements

Reference	Requirement	Action needed
D1	Game shall be aesthetically pleasing	There needs to be cooperation during asset, models and board creation to ensure cohesion with style and colours
D2	Game should be playable by young children	Simple language and how to create intuitive controls will have to be discussed
D3		

4th meeting: Wed, March X, 14:00

Attendees: Steven, Greg, Chris, David.
We fight against Unity and Git.

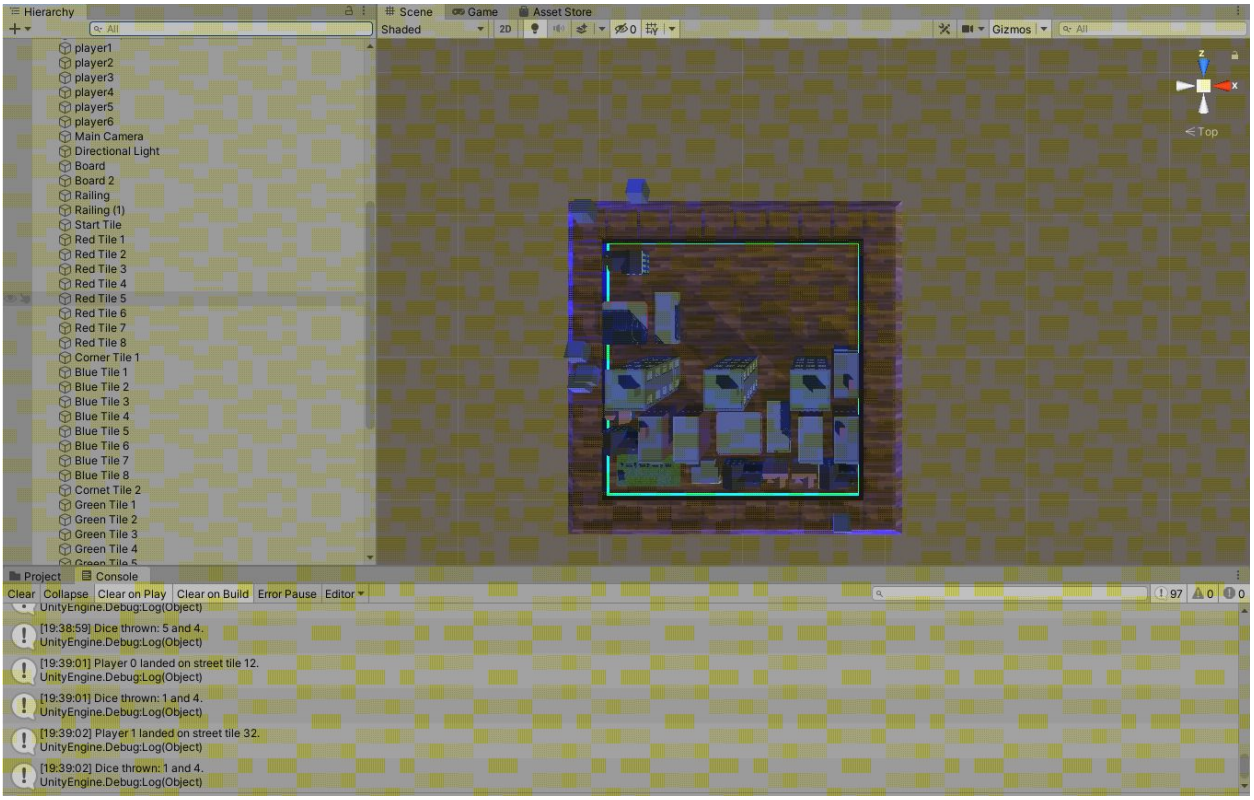
5th meeting: Wed, March 11, 14:00

Attendees

- Greg
- Chris
- David (keeping minutes)
- Steven had to leave for China but can contribute remotely.

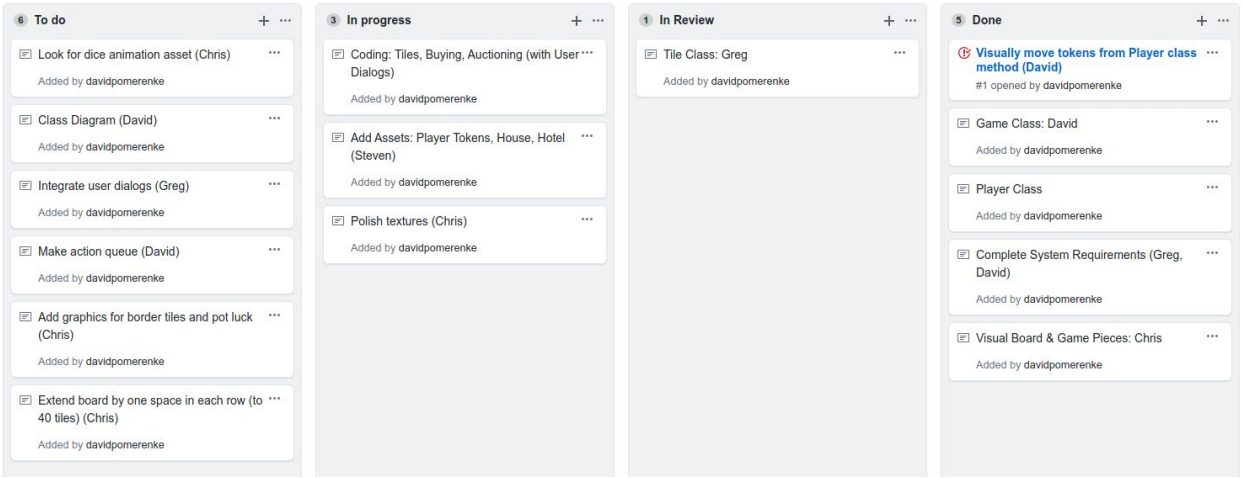
Review of progress

We now have our stuff in a Git repository. Chris improved the board design and added fancy stuff, Greg made a user dialog for buying houses (still to be integrated with the game), DAvid made a player movement animation.



Plan for next sprint

We create new task cards:



April / May

We have continued the project without further meetings. We did not proceed with weekly sprints but rather work asynchronously, communicating via Slack whenever necessary. As we all had assignments due in April, we did most of the work in May.

What we did:

- Greg coded the dialog system, also containing most of the logic of the game. He also created a documentation document.
- Chris corrected the size of the board and improved the graphics further. He also made a logo.
- Umar finally joined in and created a new Gantt chart, a sequence diagram, and a risk assessment.
- David also did some coding, reading the file data and displaying the tiles.

Issues we ran into:

- Problems with Unity + Git: We found it really hard to merge our work. A special problem was that Unity stores all objects that are created via the user interface in a single non-human-readable file, and coding also requires the use of these objects. We thought about switching to Unity Teams (as that would now have been possible with fewer team members) but ran into technical issues with that, so kept with Git. Our solution was to always communicate when we were going to change anything in that file (scene.unity) and to push these changes to master immediately, so that could directly continue working on top of these changes, rather than merging later.
  If we'd do this project again, we might just use Unity Teams, since it turned out that it is not actually necessary that everyone has direct access to the code.

- Due to the pandemic, Steven went back to China, where he appears not to have had proper internet access, so we continued working without him.

The final state of the project:

- Game-play functionality is working, including all the important stuff like throwing a dice, buying property, and auctioning. The tile data are read from the configuration file.
- Full documentation is available. Some documentation is in the code comments and most is in the Code-Documentation.pdf file.

- Pot Luck and Opportunity Knock cards are not implemented. There is a dummy class (TileDraw.cs) indicating how they could be integrated into the game.

- We have only done manual integration testing. There is no unit testing.