Mini Adventure Capitalist

==========
Author
==========
Tao Wang
iamwangtao@gmail.com

==========
Introduction
==========
The project goal is to implement a simple clone of an idle business sim-game: Adventure Capitalist.

========================
Game feature requirements
========================
1. Buy and upgrade businesses to make profits.
2. Manually start a business' producing cycle.
3. Hire managers to automatic the producing cycle.
4. Businesses continue to make progress while the game is closed.

================
Design Principles
================
The project implementation follows two principles:
1. Data-driven UI.
Minimize UI definition (layout, behavior, etc.) inside the front-end code. Load UI definitions at run time.

2. Programming to interfaces
Programming to interfaces provides a clean way to separate the declaration of functionality from the implementation.

================
The implementation
================

1. Programing language:
TypeScript was selected as the programming language due to its IDE friendly and ability to reduce unanticipated runtime bugs.

2. Dependencies:
The only code dependency in the project is the PixiJS rendering engine.
All other development dependencies can be found in the package.json.

3. Game features
All game features in the requirements have been implemented.

There are two enhanced features were added through a splash screen:
1). A player can choose between two game UI layouts before loading the main game.
2). A player can choose to reset the game to a fresh start by checking a checkbox before loading the main game.

An outstanding feature requirement of the game is “Businesses continue to make progress while the game is closed.”.
To achieve the goal, the game app needs to keep saving game states out of the game’s memory.
There are three solutions: web browsers’ local storage, a server end app, or a combination of the two. The local storage solution was selected due to the limited timeframe, and it turned out well.

4. The UI:
The UI Programming in the project could have been straight-forwarded: to write an interactive class that holds icons, texts, etc., group them together as a “business” item class, then create a list of instances of the items at runtime, feed them data for each “business”.

However, according to the principle of data-driven UI mentioned in the section of “Design Principles”, the implementation of the game UI went to a more challenging approach: a simple experimental data-driven UI codebase was created from the ground up. The interactive class and the business item class became empty content classes which will be fed with UI definitions and game data through server-side JSON files until the classes are instanced. And they will handle the game state changes through a “updateStates” public method. Predefined game states and corresponding UI reactions also came with the UI definition file.

The outcomes of the data-driven UI approach are promising:
1). The game is A/B test ready out of the box: loading different UI definition files could result in a very different UI/UX experience. The game provides a splash screen to simulate an A/B test scene: a player has choices to load the game from two predefined UI definitions: “Game layout A” and “Game layout B”. The latter is

considered a more polished game UI layout. The game UI could be changed through UI definition files only, and more UI layout definitions could be created. All these UI changes do not need a single-line change in the game app code.
2). Most classes in the data-driven code base could be used for other games/apps without code changes.

Trade-offs of the data-driven UI approach are listed below:
1). Filling the UI definition data in JSON format by hand was quite time-consuming, it proves that a real-world data-driven UI framework always needs its corresponding part: a data-producing/reviewing tool.
2). Because the UI definition is from runtime data, more data verification code and error handling code are needed.
3). The code in the data-driven UI codebase is more abstract than traditional UI code, so it's learning curve is steeper than the traditional UI code for code maintainers.

The experimental UI data-driven code base is not a full-featured framework, it can only fulfill the basic needs for the project's scale.

5. The UX:
The game UX design is focusing on visual-driven, users will see immediately the highlighted UI elements when an operation is allowed (mostly triggered by fund changes). UI elements that are not needed anymore will be hidden. Icon elements' sizes and positions also explained the game's progress. As mentioned above, those elements' behaviors are defined in the UI definition file and can be changed without republishing front-end code.

6. The coding:
1). The project was implemented as a front-end app that will load two UI definition files and a game data file.

2). By following the programming to interfaces principle, all of the game states storage functions were gathered in an I/O module and could be replaced by another I/O module if some of the storage needs to be switched to server-side storage.

3). A simple event bus singleton module was implemented to provide an event mechanism for controllers to view direction decoupling.

4). Most public functions and methods were commented on. Other functions and methods either self-explained by naming or implemented privately for simple purposes.

==========
The hosting URL
==========
Play the game:
https://taowang-ca.github.io/adc/

Readme:
https://taowang-ca.github.io/adc/docs/tao_adc_readme.pdf

Source code:
https://taowang-ca.github.io/adc/docs/tao_adc_sc.zip

Deploying package:
https://taowang-ca.github.io/adc/docs/tao_adc_hosting.zip

==========
Setup
==========
Setup for development:
$ npm install

Compile for deploying:
$ npm run build

==========
The credits
==========
Credit for the “free for commercial use” icons and background used in the game:
Designer: Haseba Studio
https://www.iconfinder.com/haseba

Designer: Hopnguyen
https://www.iconfinder.com/Mr.hopnguyen

Designer: Ruma Entertainment
https://www.iconfinder.com/oxy-Nation

Designer: Pngmagic
http://www.pngmagic.com/

Free online “Sprite Sheet Packer”:
https://www.codeandweb.com/free-sprite-sheet-packer