Chapter 1

Programmer's manual

This manual talks about the structure of the CyberCraft folder as well as the JS code of it. For installation of the game, please refer to the first section of User manual. For information about non-code part, especially if you are non-programmers, please refer to Successor's manual in the next chapter.

1.1 Game scenes

The game majorly has 12 scenes (see figure 1.1).

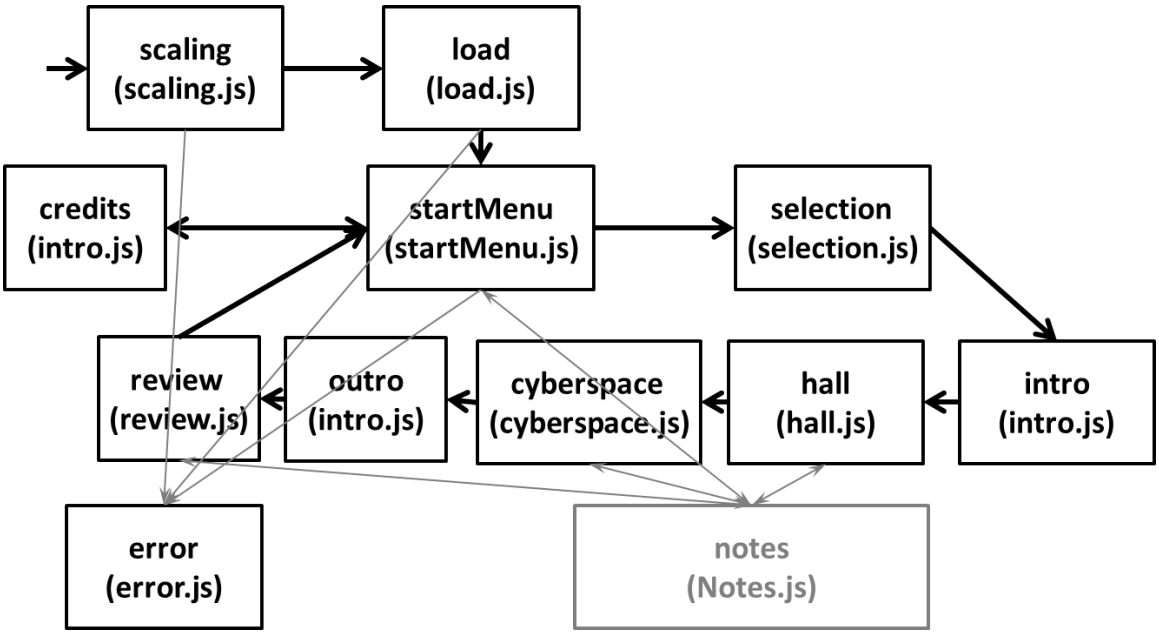

Figure 1.1. The scenes of the game

The game starts with scaling (adapt to screen size and prepare loading), pass through to load, and then reach the main menu called startMenu. For a normal game procedure, the player starts at intro (general briefing), goes to hall (specific mission briefing and hints by talking to NPCs), and enter cyberspace, where the cyber fight takes place. After the fight, it goes to outro (consequence briefing), and then review (revise which action or reaction was correct and which one was wrong), and return to startMenu. In the presence of fatal errors at scaling, load or startMenu scene, it will switch to error scene. An fatal error may be caused by one or more corrupted data files. However, for non-critical errors, including the errors with data files, a error message will pop out, but the game can still go on, without going to the error secne. All the scenes except for notes are game states, and they are defined under js/states (but the three scenes intro, outro and credits share the single file intro.js). The notes scene is not a state, because the players cannot afford the state change when they want to refer to notes during the cyber fight (in cyberspace scene). Thereby, notes.js is actually an part of modules (under js/modules).

1.2 Structure of the game folder

The game folder of CyberCraft contains the following things:

assets: This folder contains all the assets used in the game, involving images, sprite sheets, sound and music.

- **images:** All the images, whether in the form of a single image or a sprite sheet.
  - **notes:** Some pictures to be used in personal notes
  - **portraits:** The portrait pictures for the characters in the hall or in the cyberspace
  - **sprites:** The sprite sheets for the characters in the hall
- **music:** long audio files used as the BGM of the game
- **sound:** short audio files used for the sound effect of the game

css: The css files related to the tools of bootstrap and alertify used in the game

jquery: The jquery code of bootstrap and alertify used in the game

js: It contains all the JavaScript codes for the game

- **modules:** The module files in charge of a certain functionality. They are used by the game states.
- **states:** The state files each is a game state.
- **bundle.js:** this is a bundle generated from all other Javscript files. index.html actually uses this file, instead of the original js files.
- **main.js:** The "root" JavaScript files. It incorporates Phaser and the state files and builds global variables.
- **parse.bat:** The BAT file (Windows) to facilitate the rebuild of bundle file from the source files. The bundle is built with the JavaScript tool browserify. Any change in the source file will not be able to affect the game until the source files are packed into the bundle file.
- **phaser.min.js:** The simplified version of Phaser, the game engine used for this game.

scenarios: The JSON and TXT files to customize each scenarios, assets and personal notes. The content of this folder is designed as player customizable, to facilitate future extension by the players. For detailed description of the files in this folder, see chapter 2

index.html: This file is the main entrance of the game, though vast majority of the work is done by JavaScript located in js folder.

1.3 States

The Phaser game engine divides a game into different states. This provides modularity to the game development. Moreover, Phaser provides lifecycle management and sprites creation or destruction for the states. A state will be a scene from the player's aspect of view. This game is divided into 9 state files, for 11 states, all placed in the folder js/states:

scaling: The first state of the game. It scales the game canvas to the window size, and prepare (a little) asserts for the incoming load state, as well as potential error state.

error: This is the state for fatal errors when loading necessary files.

load: This state load all the game assets, as well as the JSON files and TXT files. It also reads saved game from browser's local storage.

starMenu: This is the first scene to be presented to the player, starting from where the BGM plays. The player can also read the personal notes here, without even starting a scenario.

selection: This is the scene for the player to choose a tutorial or a scenario to play. For scenario selection, the scenarios are organized in pairs, and can be unlocked in pairs, when the previous pair is completed.

intro: The file intro.js is made to manage three states at the same time: intro, outro and credits. Intro is the first scene for each scenario (except that some scenarios can have no intro). Intro tells the general information about the story setting, like when, where and who. Outro is the scene after the cyber battle, to depict the result of the cyber battle. A score will also be given to evaluate how the player has performed. Credits is the general information about the game resources. The content of intro, outro and credits are all specified in JSON and TXT files under scenarios folder.

hall: Hall is the scene after intro scene. There the player will get a better briefing of the task, in the form of a dialogue. The player will also get more information about the incoming fight, even some hints from other NPCs. The content in the hall is specified by scenarioX_NPCs.json (where X is the scenario number).

cyberspace: This is where the cyber fight commences. It is configured by scenarioX_cyber.json.

review: This is the last scene of a scenario, after outro scene. In review, the player revises his/her performance in the last fight, recognize where he/she did right and where wrongly.

1.4 Modules

Some functionality are shared between different states, some functionalities are cross referenced by other functionalities. To facilitate reuse and enhance modularity, each functionality is made

into one JS file under js/modules folder, majorly in object oriented model (they are classes), whose names starts with capital letters.

Act: A value class for the Act. An act is one operation that the character can perform in order to achieve victory. The definition of an act involves many properties, some are compulsory and some has default values.

ActManager: Being one of the "Manager", actManager is the one that creates, stores, converts acts, as well as managing the learning or application of acts. Those effects from the acts like taking damage, act failure, enforcing or cleaning buffs are all managed by the ActManager. One instance is created at the start of cyberspace.

AIManager: It managers all the AI's operations like learning or applying of acts, based on the AI script written in the scenario's cyber file. AI will randomly choose an action pattern, which can be a chain of multiple acts. When it's done, AIManager will also end AI's turn. One instance is created at the start of cyberspace

AjaxFileReader: The module that exploits Ajax to load the customizable files. It's used by load state for the loading of all the JSON and TXT files except assetsTable.json (it's loaded without ajax, before load scene).

AudioManager: The manager for all the sound effect and BGMs across the scenes. It's created during load state, and alive for the whole process of the game.

BuffManager: It manages the buff database as well as the buffs on the characters. Note that the influence of the buffs on the application of an act is managed by ActManager. One instance is created at the start of cyberspace.

dynamic_text: It's a module that provides the typing-like animation for the text of the dialogues as well as that of intro and outro and some of credits.

EffectManager: It manages the major animations in cyberspace like attacking, defending, Popup words, unhappy face. One instance is created at the start of cyberspace.

GameManager: It manages a more general things of the game in cyberspace, like income, resources, server assets, round initialization. One instance is created at the start of cyberspace.

HintBox: It provides the hint box across the states, to display a hint when the players mouse hover over a clickable button. One instance is created for each state that need it.
Used by: startMenu, hall, cyberspace, review.

learning_data send: This module sends the player's learning data to the data collection server, so that information about the game's influence on the player can be obtained.

loadSave: A module managing the loading and saving of game data into the local storage of the browser.

LogEntry: A value class for one piece of log for the characters' performing an act, how it goes, and, if failed, why it failed.

LogViewer: It manages the display of action logs by cyberspace as well as review state. One instance is created at the start of these two states.
Used by: cyberspace: action log, review.

MultimediaText: It manages the creation of pure text, text with typing-like animation, dialogue with typing-like animation and image-text mixed page. This class is reference by multiple states and classes.
Used by: notes, credits, intro, hall: dialogues, cyberspace: in-game dialogues, outro.

Notes: It build up the personal notes interface for the player to read. Note that although the personal notes is opened full screen, it is not designed as a state. It' only an upper layer over the original layer, so that it only "pause" the current state. In this way, the player can easily return to where they were before opening the personal notes. Otherwise, the player may think twice before opening personal notes, for fear that he/she would loss the current progress. One instance is created for the startMenu, hall, cyberspace and review, to make the personal notes as accessible as possible.
Used by: startMenu, hall, cyberspace, review.

NPCManager: It manages the dialogues of the NPCs in the hall. The NPCs has their states, which indicate which dialogue the NPC will say when he/she is clicked. The dialogue has also dependencies. e.g. NPC1 will not say dialogue 2 before NPC2 has said dialogue1. This functionality is suitable for designing a story like tutorial 1. The configuration of the dialogues is written in scenarioX_NPCs.json One instance of NPCManager is created at the start of hall state.

PersonalNotes: The value class for personal notes. One instance is created by loading, and remains for the whole game.

ScriptManager: It manages the script as written in "scripts" of the scenario's cyber file. A piece of script will be active at the start of a certain round, displaying dialogue and unlocking new acts for the characters. It's an excellent tool for the tutorial or the first scenarios, in the sense that the player will see the acts unlocked progressively, and he/she will get in-game hints and guidance. Both these two properties conform to the guidelines for game tutorials.

ScrollButtons: It provides the scroll up and scroll down buttons and the indicator for the current page among the all available pages. Nevertheless, the displaying of the actual content of the page is managed by the caller function. After all, all the multipage interfaces use this class, but the way to display the content of the page diverges a lot.
Used by: notes, credits, selection, intro, cyberspace: acts on the main panel, cyberspace: buffs, cyberspace: action log, outro, review.

tile_functions: This module is used when creating the scene for the hall, from the tiled map file hallMapIntruder.json and hallMapDefender.json.

Chapter 2

Successor's manual

The game supports easy extension of the scenarios, even by non-programmers. The successor's manual concentrates on the content of these extendable files as well as the data format that should be complied when creating new scenario files.

The scenario files are in TXT and JSON format. TXT files can be edited with any text editor. The JSON files can be added with text editors also, but if a tool with more support like word coloring, bracket pairing, things can be easier. One of the possible tools is notepad++, which support a large variety of data format and programming languages alike. Alternatively, online JSON editors are also a good choice for the work of JSON files specifically. One of the online JSON editors locates at http://jsoneditoronline.org/.

2.1 File structure

To add new scenarios, one needs to add files to the scenarios folder. The folder contains the following files (most of the files are bind to a particular scenario. Therefore, the scenario number in the filename will be represented as an X here):

- README.txt: It's a text file describing the content of this folder, similar to what is written here. Not expected to be changed.
- hallMapIntruder.json and hallMapDefender.json: A json file made with Tiled to illustrate the 2D map of the hall. If you want to create a better background for the hall scene, e.g. adding more decorations, you can do it here. However, if you want to add any moving objects or animations, you have to touch the code in js/states/hall.js. If you want to add new assets to the hall, you have to add one mapping to assetsTable.json and add one statement to js/states/hall.js (you should recreate the bundle before any change to the JS files takes effect).
- personalNotes.json: This file stores the entries related to the personal notes. Most importantly, all the acts and buffs activated in the game will have a link to the corresponding entry in the personal notes with exactly the same name. The personal notes will then discuss the security terms more deeply. It's strongly suggested that, whenever a new act or a new buff is added to a scenario, an entry of it to be added to personal notes also.

- credits.txt: What will be shown at the credits scene. Add credit to you for your contribution to the game.
- assetsTable.json: An table for all the images, sprite sheets and audio(sound and music), to be loaded at load scene, which covers nearly all the assets of the game. images contains the path of the image and the key assigned to it. spritesheets need also the specification of frameWidth and frameHeight. For audios, urls takes the value of a single string or an array of strings. In this game, all urls are given in single strings though.
- common_acts.json: The common acts file defines the acts and buffs used by cyberspace scene that are frequently shared across scenarios. However, for a certain scenario, not all of the acts or buffs are used. Also, the scenario can define its own acts or buffs in addition to those defined here. The scenario can even redefine some acts or buffs. All these settings for the scenario are done by the files called scenarioX_cyber.json
- scenarioX_cyber.json: The cyber file specifies the data used in the cyberspace scene for scenario X exclusively. Examples of the containing data include the player's role in the fight, the defender's assets, as well as the acts and buffs used in the cyberspace. Note that, this file takes some acts or buffs from common_acts.json, but it also define its own acts or buffs.
- scenarioX_intro.txt: (optional for each scenario) The intro file illustrates what will be shown at the introduction scene of a scenario. It involves texts as well as pictures. With it the players are supposed to see a short introduction of the story of this scenario before the game formally starts (e.g. a short briefing of time, place and characters).
- scenarioX_NPCs.json: The NPC file stores for the scenario the information of the NPCs that will be found in the hall. Most important of all, it contains the dialogues of the NPCs, as well as how the NPCs change their dialogues (states) with the player's interaction with them. The hall is supposed to be a place where the player can freely gather information related to the incoming fight. The NPCs are to give hints and recommendations for the player.
- scenarioX_outro.txt: (optional for each scenario) The outro file is similar to the intro file, only that it's put at the end of the level, showing the result of the what the player has done in the game. It also support texts and pictures. It's a good place for the revision or extension of the knowledge delivered in the game.
- tutorial2_cyber.json and tutorial3_cyber.json: The cyber file for the two tutorials teaching things about cyberspace. Tutorial2_cyber teaches how to play as defender, and tutorial3_cyber teaches how to play as intruder.
- tutorialX_intro.txt: The tutorial equivalence of introduction file.
- tutorial1_NPCs.json: The NPC file for tutorial 1 exclusively. Only this tutorial focuses on hall scene, so only tutorial 1 has NPC file.
- tutorialX_outro.txt: The outro file for the three tutorials.

Therefore, to add a new scenario, one needs to add one cyber file and one NPC file, and optionally an intro file and one outro file. The scenario number is not in any way bound to the content of the scenario. Thereby, you can renumber scenarios as you want. By doing so, you can even adding new scenarios in between old ones. However, the scenario numbers always have to be consecutive. This is because the loader stops searching at the first missing scenario. It's also recommended to add entries in personalNotes.json for every new acts and buffs you have created.

2.2 File format details

1. personalNotes.json: the note have a first entry named "Personal notes", which tells something in general about the note itself. The root element "desc" here is just that "something about itself".
   "acts" contains "offDesc" and "defDesc", which are the descriptions of offensive and defensive acts respectively. "offDesc" and "defDesc" are followed by "offensive" and "defensive", which defines the entries of offensive acts or defensive acts. Similarly, parallel to "acts", there is also "buffs" with "buffDesc", and under the other "buffs" element of it, the entries for the buffs are defined. The definition for offensive acts, defensive acts or buffs follows this format:

   (a) name: A name of the security term (the entry). Acts and buffs of exactly the same name will obtain a link directly to this entry.

   (b) desc: the description of it. The description can contain multiple pages, delimited by the caret character (' ^ ').

   `e.g. Page1.^Page2 sentence1. Page 2 sentence2.^Page3.`

   Each page can be a pure text page or an image-text-mixed page. The pattern is quite similar to that of the intro file described below, except that the pure text will not be displayed with typing animation.

   (c) sees: (optional)An array of links to other entries in the personal notes. It's recommended to try to add some links to related security terms, so that a knowledge graph is built.

   (d) url1 and url2: (optional)Links to external websites where the security term is further explained. The link to external resources in addition to the personal notes itself, serves as a more professional resource for the players with more motivation and more curiosity.

   It's recommended to create an entry for each new acts or buffs defined in common_acts.json and scenarioX_cyber.json (with exactly the same name), so that the players can always have a place to refer to when they get confused with the term.

2. scenarioX_intro.txt/scenarioX_outro.txt/tutorialX_intro.txt/tutorialX_outro.txt/credits.txt: these files follow the same pattern: It involves pages of pure text or image-text-mixed pages. The pure text page will be displayed with typing animation, while the image-text-mixed page will be displayed immediately. The pages will be delimited by a caret character (' ^ ').

   A pure text page contains only pure text of course. Note that the caret character (' ^ ') is reserved character, which is not supposed to be anywhere inside the page. Character ' # ' is also not supposed to be placed at the start of the pure text page.

   An image-text-mixed page is characterized by a sharp character (' # ') at the start of the page. As a matter of fact, this kind of page is composed of images and texts, each described after a ' # ' character. Here, newline as \r, \n or \r\n will be ignored by the program.

   To create an image, follow this pattern:

   `#image$<x coordinate>$<y coordinate>$<path to the image file>$<width>$<height>`

   width and height are optional. Assigning both values will shrunk or enlarge the image as you want. However, if you specify only width, height will take the same value, resulting in a square image.

   To create a text in the image-text-mixed page, follow this pattern:

```
#text$<x coordinate>$<y coordinate>$<text to create>
```

The text can also have one more argument:

```
#text$<x coordinate>$<y coordinate>\$<text to create>\$<word wrap width>
```

Note here that, all those within angular brackets should be replaced with actual values. Also, if the readers know about Phaser, they will realize that the arguments in this pattern are just aligned with the arguments to define image sprite and text sprite.

3. scenarioX_NPCs.json:
The file contains an array called NPCs, whose elements are description of each NPC in the scenario.

   (a) name: the name of the NPC. One can also add the title and the profession can.

   (b) sprite: the path to the picture of the sprite of the NPC

   (c) portrait: the path to the picture of the portrait of the NPC

   (d) x: the x coordinate to put the NPC sprite in the game

   (e) y: the y coordinate to put the NPC sprite in the game

   (f) speeches: the array of the NPC's speeches. A speech is a chain of sentences (potentially multi page) that the NPC will say without stoping. Though not specified here, when the game runs each NPC will have a state. e.g. at state 0, when talked to, the NPC will use the first piece of speech; at state 1, the second piece of speech

   Each piece of speech contains prerequisites (optional) and speech. speech is of course what the NPC will say at the particular state. Long speech need to be divided into multiple pages. The character ' ^ ' will be used to separate pages.
   prerequisites is the condition on other NPCs' states that need to be firstly meet, before this NPC will come to this state. This functionality is useful to build an experience of discovery in the hall scene. e.g. NPC1 asks the player to collect information by asking NPC2. NPC1 will repeat the same sentence if the player just keeps clicking on NPC1. When the player has talked to NPC2, having found the information, setting NPC2's state to the appropriate value, NPC1 will change to the next state to congratulate the player for the finding. To see a really definition, if the prerequisites is the following:

```
"prerequisites":
[
 {"npc": "npc2",
  "state": 2},
 {"npc": "npc3",
  "state": 3}
]
```

   That means npc3 has to reach state 2 (the third speech defined in speeches), and npc3 has to reach state 3, before this npc will come to this particular state.

   There is one special speech (state), which is the speech asking if the player is done with the talk in the hall, and is ready to go into the cyberspace to challenge the opponent. This particular speech is necessary for every scenario (in order to process from hall scene), and it's indicated by a grave accent (on the tilde key adjacent to number 1 in America keyboard)

at the first page of a speech. When an NPC when talked to, reaches this speech, a question like "Are you ready for the cyber battle?", together with "Yes" button and "No" button will be shown. The real question can be configured by writing a single page text after the grave accent. Extra pages of text after the grave accent will be ignored by the program. N.B. don't add more speeches (states) after this final question. If you do so, if the player clicked on the "No" button, the NPC will never ask the question again, trapping the player forever in the hall. Final point: the first NPC will come to talk to the player when the player enters the hall, even if the player has not clicked on anyone. It's quite reasonable for this NPC to be the manager or the desk clerk of this place

4. common_acts.json:

   (a) acts: it consists of two arrays. The first array defines acts for the intruder, and the second array defines acts for the defender.
      i. name: (required)the name of the act
      ii. prerequists: an array of acts that need to be unlocked before unlocking this act
      iii. learningCost: the cost to unlock this act
      iv. desc: short, single page description. It's shown in non-scrollable popup window, so don't make it long. To put long explanation, use personal notes
      v. needSelfBuffs: the player has to have this buff in order to perform the act
      vi. needRivalBuffs: the rival has to have these buffs for the player to perform the act successfully
      vii. noSelfBuffs: the player should not have these buffs in order to perform the act
      viii. noRivalBuffs: the rival should not have these buffs for the player to perform the act successfully
      ix. cost: (required) the cost of performing this act. Zero-cost act cannot be performed. It will be used as prerequisite for other acts.
      x. successRate: the initial success rate of the act. It can be modified in game by other acts. By default, it takes 1.
      xi. selfBuffs: the buff enforced to the player when the act succeeds
      xii. rivalBuffs: the buff enforced to the rival when the act succeeds
      xiii. cleanSelfBuffs: the buff cleaned on the player when the act succeeds
      xiv. cleanRivalBuffs: the buff cleaned on the rival when the act succeeds
      xv. buffLength: for how many rounds the buff will remain on the player/rival. Setting buffLength as -1 means the buff is has infinite length; 0 is erroneous; 1 means the buff expires at the end of your round; 2 at the end of rival's round. For majority of the cases, a positive even number is expected. Default to -1.
      xvi. bonus: the bounty for the intruder when the act succeeds. It's also the damage to the assets of the defender
      xvii. spamRequests: the spamRequests generated from the buffs of this act (it's normally supposed to be an offensive act enforcing one single buff to the defender). Default to 0.
      xviii. modifier: a string which modifies some properties of other acts.
      modifier format: The string can contain multiple modifications, each delimitered by the semicolon (' ; '). For each modification, it contain five parts delimitered by colon(' : ')
      <role>:<act name>:<property>:<operant>:<amount>

role==0 means it modifies for the offensive acts, while role ==1 means it modifies for the defensive acts. property takes values like successRate or spamRequests. operant takes one of the five values: '+', '-', '*', '/', '='. They represent assignment operation '+=', '-=', '*=', '/=', '=' respectively.

xix. learnt: if the act is initially learnt at the start of the game. Default false.

xx. unlocked: if the act is already unlocked at the start of the game. Default to true. Acts Initially locked is supposed to be unlocked by scripts.

(b) buffs: it's an array of buff definitions. The definition contains the following values:

i. name: (required)the name of the buff. It's possible to use the same name as the act which enforces the buff.

ii. desc: (required)the buff description. It's also a single-page short description within a non-scrollable popup window, so don't make it long. To put long explanation, use personal notes.

iii. capacity: the extra server capacity provided by this buff. Default to 0.

iv. upkeep: the amount of resource lost each defender's round, due to the presence of the buff. Default to 0.

v. dosResistance: the resistance to DoS attacks provided by each buffs. Default to 0.

5. scenarioX_cyber.json and tutorialX_cyber.json:

(a) name: the name of the scenario

(b) serverCapacity: the initial server capacity of server to serve incomming requests

(c) serverAccessValley: the minimum amount of requests to the server each defender's round

(d) serverAccessPeak: the maximum amount of requests to the server each defender's round

(e) initialResource: how much resource the intruder and the defender have (before round 1)

(f) constantIncome: the resource obtained each time the character starts his round. This guarantees a minimum income. However, it's more suitable to let the defender gain resource from responding to the client requests rather than from this.

(g) maxResource: the players can't keep more resource than this value.

(h) maxRounds: the number of rounds of this scenario. If the defender can sustain until this time, the defender wins.

(i) assets: the initial "HP" of the defender. The defender will be penalized on assets at each credential breach. If the assets dropped to zero or less than zero before the maxRounds reaches, the intruder wins.

(j) defensive: if true, the player plays as the intruder; if false, the player plays as the defender. The AI written in the latter part of this file should be consistent with the value here: the AI plays the opposite role.

(k) characterName: the name to be displayed as the name of the intruder and the defender

(l) portrait: key value of the portrait pictures of the intruder and the defender

(m) commonActs/commonBuffs: the name of the acts/buffs activated in this scenario, which is defined in common_acts.json. If this scenario will modify any parameters of an act/buff, the act/buff should not be listed here.

(n) acts/buffs: the new definition and redefinition of acts/buffs for this scenario, compared to those in common_acts.json. For detailed format of definition, refer to common_acts.json above.

(o) initialBuffs: describes the buffs that the intruder or the defender already have at the start of the game. name: the name of the buff. length: for how many rounds the buff will last. length == -1 means the buff will be there from the start till the end; length >0 means the buff will be there for the first rounds; length == 0 is erroneous.

(p) AI: an array of action patterns that AI will follow.

i. pattern: an array of acts characterizes this action pattern. When the pattern is chosen, AI will firstly guarantee the acts are all learnt (could take multiple rounds). Then, if the resource is enough, AI will apply these acts in sequence in one round. If the resource is insufficient for all the acts, AI will wait, until the necessary resource is obtained. The acts in the pattern are usually supposed to be a combo.

ii. chance: the chance that AI will choose this pattern out of all others. Note that if any of the acts in the pattern is not unlocked yet, the pattern is considered lock, and AI will ignore the pattern.

The sum of the percentages of all the action patterns can exceed 1. This is because for each turn, AI will only consider the unlocked patterns. The sum of unlocked action patterns can also exceeds 1. In this case, the patterns will consume the chances in sequence, meaning that the last patterns may suffer their chance decreased. This is understandable and even useful, in the sense that one can make use of this effect to reduce AI's likely hood to perform some acts that are designed for the earlier stages.

(q) scripts: It's an array of scripts, each following this format:

i. round: the script will be activated at the start of this round

ii. dialogues: the dialogues to display:

- name: the name to be displayed for the speaker of the dialogue
- portrait: the portrait of the speacker
- dialogue: the (multi-page)dialogue to display. Use ' ^ ' to delimit pages.

iii. newActs: arrays of acts to be unlocked. First array for the intruder and second array for the defender

iv. shouldApply: (should be specified only for player's rounds) it's an array of act names. It enforces the player to apply those acts in the designated round. If the player has not applied all the acts in the round, the "End turn" button will be locked. This functionality is useful for the tutorial section, to force the player try out newly taught acts. Nevertheless, this functionality reduces player's freedom, so it's deprecated for the formal scenarios. Also be aware, when adding this constraint, one should guarantee those acts are already unlocked, and the resources are adequate. Otherwise, the player will fall into a deadlock.

About numbering of scenarios: one can create as many new scenarios as he/she want. If new scenario is to be put after the old ones, just number them increasingly. To add scenarios in the middle, however, one has to renumber those old scenarios, leaving a gap for the new one. In either case, remember that the program will stop searching for more scenarios at the first absent number, so, do number them continuously.

2.3 Error messages

If you modify the files in the scenario folder, you may encounter some error messages related to the inconsistence or wrong format. Here are some of the erros messages (XX, YY or ZZ are going to be replaced with the values based on your actual situations):

- **Cannot find scenarios/assetsTable.json or the file is corrupted!**
  Maybe assertsTable.json is missing; maybe the file does not conform with JSON format. One common mistake is missing or having too much commas in an array.
- **Error! The act "XXX" activated for this scenario (scenario Y) is not defined in common_acts.json!**
  In scenarioY_cyber.json, under the element of "commonActs", you have written XXX. This means the scenario will take the act definition for XXX from common_acts.json. However, it's not found. Maybe it's because of wrong spelling (the name is case sensitive and should match exactly). Maybe you have got a wrong role: intruder's acts are always in the first array, while defender's acts always in the second array.
- **Error! The prerequisite "ZZZ" of act "YYY" activated for this scenario (scenario Z) is wrong!**
  In the definition of act called YYY from scenarioZ_cyber.json, the property of prerequisites is specified. However, at least one of the prerequisites (should be an act name) is not an act defined for this scenario.
- **Error! An act name is missing Recheck scenarioX_cyber.json**
  In the definition of an act, the name is compulsory
- **Error! The act "YYY" is missing cost Recheck scenarioX_cyber.json or common_acts.json**
  In the definition of an act, the cost is compulsory. No act should cost zero resource. Zero-cost act on the other hand means the act cannot be used. Then the act is only put in the game as a prerequisite for other acts.
- **Error! The buff "XXX" related to an act in scenario Y is not found!**
  This scenario will enforce/clean buff on the character. Nevertheless, the definition of the buff is not found.
- **modifier format wrong!**
  It's not suggested to use modifier if you have other alternatives. But if you do want to use it, the modifier should be written as a string, delimited by semicolon (;) into substrings, each following this regular expression:

  ```
  ^[01]:[^:]*:[^:]*:["+""-""*""/""="]:[0-9]+\.?[0-9]*$
  ```

  For detailed description, see "File format details" above.
- **Error! The act "XXX" specified in the AI pattern for this scenario (scenario Y) is not defined!**
  Under the element pattern of AI, you have given a name that is not seen as an act name.
- **Warning! The buff "XXX" activated for this scenariol (scenario Y) misses description**
  A buff called XXX is not defined. The program assumes the buff has empty description and no other properties (capacity, upkeep, dosResistance). But it's still recommended that you define the buff or import it from generic_acts.json.

- **Sorry. This entry is not found in personal notes!**
  It's recommended that when you add new acts or buffs to the game, you add entries to the personal notes also. In this way, the player will have a in-game help about the new security term. Also, personal notes serves as an in-game source of knowledge for those interested to learn.

- **Error! in the scenario file scenarioX_NPCs.json, in the prerequisites of the speeches, a reference to a npc name is not found!**
  The dependency of dialogues is based on the name of the NPC. Probably you have changed the name of the NPC, but have not changed the references to it. Remember that it's case sensitive.

- **Warning! The script for round X is not understandable.**
  The script will run at the start of round X. X has to be a positive number of course.

- **The game just stop at loading or during the game**
  It's quite possible that there are other runtime errors that can be viewed only by checking the console. The way to open console could be different from browser to browser, but the console is usually one part of Developer tools (opened with F12). For messages at console, see below.

There are also some other console messages that can be seen only by opening developer tools (e.g. with F12 key):

- **GET http://localhost/CyberCraft/scenarios/scenarioX_cyber.json 404 (Not Found)**
  This message always comes. The game gets to know about the number of scenarios only at the first cyber file not found. That is to say, the game has found X-1 scenarios. Therefore, there is no problem here.

- **scenarioX_intro.txt not defined/scenarioX_outro.txt not defined**
  You have created a scenarioX_cyber.json, which indicates a new scenario is added. However, the intro or outro file for the scenario is not created with it. There is no problem if you don't have intro or outro files. The game will just skip them.

- **scenarioX_NPCs.json not defined**
  The NPC file is always needed. The absence of NPC file will result in an error sooner or later. If you don't know what to be written in the NPC file, just put a single person, with the only dialogue leading to the cyberspace (a single page dialogue starting with the grave accent character).

- **Error loading asset from URL assets/XXX/XXX.png**
  (This is a warning) You have added new reference to assets in assetsTable.json. However, the path is wrong.

- **Uncaught SyntaxError: Unexpected token , in JSON at position XXX**
  One of the JSON files has syntax error. It could be the newly added JSON file. A debugger may offer you a link directly to the place of the error.