Project Documentation

Folder Structure

- Core
- Game
- Scene
- Store
- Common
- AssetsLoader
- Events

-Core

Installer

Abstract class responsible for installing game classes.

Example usage:

```
export class SceneInstaller extends Installer {
   install(): void {
      this.container.bind("SceneManager", new SceneManager(this.app, this.container));
   }
}
```

DIContainer

A class responsible for binding classes added through installers. This allows accessing classes anywhere in the code via:

```
this.sceneManager = DI.container.resolve<SceneManager>("SceneManager");
```

Di

This class manages instances of **DIContainer** and **app:Application**.

MVC

The MVC pattern is used for creating View, Model, and Controller.

In Controller, we add dependencies (Model and View), initialize them, and create classes.

Example:

```
this.aceOfShadowsScreenController = new AceOfShadowsScreenController(
   new AceOfShadowsScreenModel(),
   new AceOfShadowsScreenView()
);
this.aceOfShadowsScreenController.init();
this.addView(this.aceOfShadowsScreenController.view);
```

Manager

Acts like an installer added to an Installer to provide access to components from different parts of the code.

BaseGame

The main entry point of the project. Responsible for creating Application, DIContainer, installing RootInstaller, and adding canvas rescaling.

-Game

Contains GameInstaller and GameManager, which start the game and transition to MainScene.

-Scene

Contains SceneInstaller and SceneManager, which control scene transitions.

```
export enum SceneType {
   MainScene,
   AceOfShadowsScene,
   MagicWordsScene,
   PhoenixFlameScreen,
   GuessCardGameScreen,
}
```

Use loadScene(key: SceneType) to switch scenes.

Scenes and their responsibilities:

- **MainScene** – Displays game buttons.
- **AceOfShadowsScene** – Handles card animations and transfers between decks.
- **MagicWordsScene** – Displays sequential dialogues.
- **PhoenixFlameScreen** – Shows fire particle effects.
- **GuessCardGameScreen** – Mini-game for guessing a card.

-Store

Stores game data.

- **GameModel** – Stores data related to resizing and visible scene points, such as left, right, top, bottom, visibleWidth, visibleHeight, scale, etc.
- **WordsModel** – Stores data for MagicWords.

-Common

Contains reusable components used throughout the project, such as **Buttons, Card, FPS counter, ParticleSystem.**

-AssetsLoader

Responsible for loading resources.

Usage:

1. **generateAssetsJson.ts** – A script that scans the assets folder and generates assets.json.
2. **registerAssets** – Registers assets for loading; installed in Game during application initialization.

AssetsLoader methods:

- **add** – Add an asset to the loading registry.
- **addGroup** – Add a group of assets to the registry.
- **load** – Load assets.
- **loadFromUrl** – Load assets from a URL.
- **loadAll** – Load all assets in the registry.
- **get** – Retrieve an asset from the cache by key.
- **has** – Check if an asset exists in the cache.

-Events

GlobalDispatcher responsible for subscribing, unsubscribing, and dispatching events globally.

-Game Flow

1. **Game** – Entry point; creates RootInstaller, DIContainer, and Application.
2. **RootInstaller** – Adds GameInstaller.
3. **GameInstaller** – Creates and adds GameManager and SceneInstaller.
4. **SceneInstaller** – Creates and adds all game scenes.
5. **GameManager** – Launches the initial game scene.
6. **SceneManager** – Manages all scenes.
7. Scenes:
    - MainScene
    - AceOfShadowsScene
    - MagicWordsScene
    - PhoenixFlameScreen
    - GuessCardGameScreen