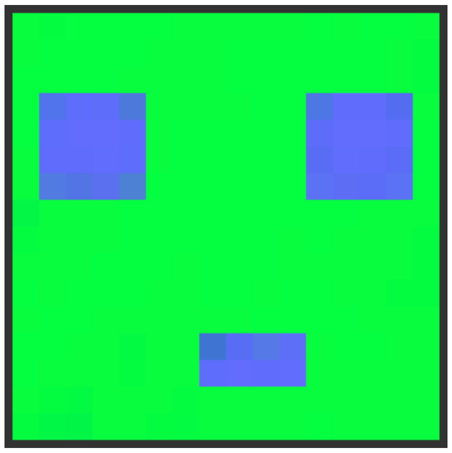
Sokoban Documentation

Brendan Lauck

November 26th, 2018



1. Introduction

This brief documentation serves as a guide to the Sokoban minigame within the Gamification project. Brendan Lauck developed this game from January 2018 to December 2018. His point of contact is brendanlauck@gmail.com and he *may* be open to answering questions.

2. User Interface

2.1 Starting up

As the user selects the Sokoban minigame from the Gamification's main menu, they enter the Sokoban main menu. From this view, the user can select from two buttons: a "PLAY" button and a "SETTINGS" button. The "PLAY" button will start gameplay. The "SETTINGS" button will bring up the game settings (note: this component is not finished).

2.2 Gameplay

The game consists of a character, walls, boxes, and crosses. The goal of the game is to maneuver the character to push the boxes onto the crosses. The character's movement can be controlled with the arrow keys. The character can move one space at a time. The boxes can only be pushed by the character. The walls act as barriers that cannot be moved onto by the character or the boxes. Each level can be reset by pressing the "R" key or by clicking the "Reset" button. This is useful for when the character gets stuck on a level. In the upper right corner of the window, the user can view the level they're currently on and a timer. Each level is timed and once the user beats the level, the time is saved for that level (note: time and level number are currently not being saved). Once a level has been completed, a “Next” button appears. The user can click this button to proceed to the next level.

The progress of the player is saved each time they finish a level (note: this is not yet implemented). As the user progresses through the game, each level increases slightly in difficulty from the last. There are over 80 levels (note: more level balancing should be done in the future). Once the user completes the last level, they are redirected to the main menu (note: this is not yet implemented).

3. Components/Resources/Textures

There are core components that make Sokoban functional. There are three scenes: MainScene, MainMenu, and LevelScene. There are four prefabs: the box, the character, the wall, and the cross. These components are used multiple times though out the game.

The MainMenu scene displays the main menu, the MainScene handles gameplay, and the LevelScene helps to build the levels by using the prefabs.

Within the Resources project folder, there is a file called ‘Levels.txt’. The Levels.txt file is used to design each level. The ‘#’ represents a wall, the ‘@‘ represents the character, the ‘$’ represents the box, the ‘.’ represents the cross, and the ‘+’ represents a character on top of a cross. The ‘;’ separates each level. The numbers are neglected.

The background art and music of the game can be replaced or customized with ease as they are located in the Resource project folder. Be aware of copyright details while selecting these files.

The Textures project folder contains the sprites for the prefabs. These were original works by Brendan Lauck, however they can still be replaced or customized.

4. Code

The code for this game is minimal. Given the simplicity of Sokoban's mechanics, there aren't any universal design patterns currently being used. There are nine scripts that are currently being used.

4.1 AudioSingleton

This script uses a singleton design pattern to play music while the user plays Sokoban. The AudioSingleton object is instantiated only once when the user enters the minigame and continues through the progression between scenes and levels. No other instantiation of this object is allowed.

4.2 SokobanMenu

This script simply loads the MainScene which displays the Sokoban main menu UI to the screen.

4.3 Levels

This script loads the Levels.txt file from the Resources project folder. It then parses this file to separate out the different levels. The user can view the Unity console while running the Sokoban program to see that levels are being added or even if there is a failure in adding a level. All levels are stored in a list to be accessed and built during gameplay.

4.4 LevelBuilder

This script uses the box, character, cross, and wall prefabs to build the levels. It accesses the level designs from the list created by the Levels script. In its current state, the levels are accessed and built sequentially (note: this script can be modified to add a stage select feature or something of the like).

4.5 Box

This script defines the properties and behavior of the box prefab.

4.6 Character

This script defines the properties and behavior of the character prefab.

4.7 LevelTextScript

This script defines the properties and behavior of the text displaying the current level that the user is on.

4.8 LevelTimerScript

This script defines the properties and behavior of the level timer.

4.9 GameManager

This script manages the flow of gameplay and keeps track of game states (note: This is a mess. Reconstructing with a game state design pattern could clean this up).

5. Testing

There is not much for this section. In the game's current state, play testing is the most efficient testing method. Running and starting up the game is quick and easy. Testing a certain level is just a matter of placing the level’s design at the beginning of the Levels.txt file (note: implementing a level select feature could make testing even quicker).