# Digital System

## Sokoban Project Report

Boxuan Song, 999025455
Yue Jiang, 999027808

## Contents



# 1. References

1. The Boxxle game on Game Boy by Thinking Rabbit, as a reference for the game design and logic. (Where I download it)
2. Resources and code from the lab.

# 2. Game Description

## 2.1. Features

- Full Sokoban mechanics, with a reset button to restart the level.
- A splash screen on start and victory screens for each level.
- A move counter showing your current moves. (Capped to and wrapping on 255)
- An undo function for the last step. (Once only)

## 2.2. How to play it

- The game starts at a splash screen showing the title. Press `START` to start.
- When a level is finished (all boxes are on the goal), press `START` to the next level. When all levels are finished, you win the game.

#### 2.2.1. Keys when playing

1. Arrow keys → Control the movement of the player;
2. Start → Restart this level;
3. Select → Undo one step, nothing to do if already undone;
4. A/B button → No any effects.

# 3. Implementation Description

In our implementation, all the tiles are located in background. We do not use objects.

## 3.1. Finite state machine

Our groups uses finite state machine to control the game state. We set 5 states for the games.

#### 3.1.1. `Splash`

A simple state that shows the title of the game. It will wait for the `START` key to be pressed, then it will change to `LevelLoad`.

#### 3.1.2. `LevelLoad`

A state that loads the level from the ROM. It will set up the initial game state, read the level map from the ROM to the VRAM (BG tile map) and scan the map for variables `man_pos` (initial address of the player tile in the VRAM) and `box_num` (numbers of boxes to be pushed into goal). After loading, it will change to `LevelPlay`.

#### 3.1.3. `LevelPlay`

##### 3.1.3.1. Ideas

We write LevelPlay based on the idea: “We only need to preserve the location of the man.” After pressing the key, we will find out whether a place is reachable or the box will be pushed.

After Readkeys, we can know in which direction where the man wants to go. Then after calculation, we can find out whether the place that the man wants to go has any objects.

1. If the place where the man wants to go is the wall, then the man will remain where he is.
2. If the place where the man wants to go is empty or the goal, then the man will directly go to the place and that’s the end.
3. If the place where the man wants to go is the box, then the man will try to push the box.
   1. If the place where the box goes is the wall or the box, then the box will not be pushed.
   2. If the place where the box goes is the space, blank or goal, then the box will be pushed to that place.

#### 3.1.3.2. Determine if the level is finished

We use variable `box_num` to preserver how many boxes that need to be pushed. There are `box_num` boxes still needs to be pushed into the goal.

1. Whenever the box enters the "goal", it will decrease the `box_num`;
2. Whenever the box leaves the "goal", it will increase the `box_num`.

Whenever the `box_num` reaches zero, it shows that all the boxes have been put into the right place.

### 3.1.4. `LevelWin`

A state that shows the victory screen, or directly change to `GameWin` if all levels are completed. If `level < LEVEL_NUM`, it will wait for the `START` key to be pressed, then it will change to `LevelLoad` to load the next level.

### 3.1.5. `GameWin`

A state that shows the victory screen of the game. Nothing to do but loading and showing the victory screen.

## 3.2. Tiles and graphics

The tiles are stored in the background, fixed at the upper left corner with $20 \times 18$ tiles. Tiles for the game objects are drawn by us using a simple tile editor, also made by us (using MS Excel). Tiles for texts are from the lab.

## 3.3. Building

The project can be built using the `make` command. The build process will:

1. First, generate the final `data.asm` from `data.nolevels.asm` and `levels.txt` by a Python script, to transform the level data into a format that can be used by the assembler;
2. Then, assemble the source code, link it and fix it to create a Game Boy ROM.

However, you can also build the project only with `main.asm`, generated `data.asm` and the "standard" `hardware.inc` by `rgbasm`. The whole project source files could be found in the attachment of this PDF.

## 3.4. Shortcomings

The position of tiles in the background is fixed and cannot be separated into smaller parts. If the player and boxes are required to move smoothly, then they should be written as objects.