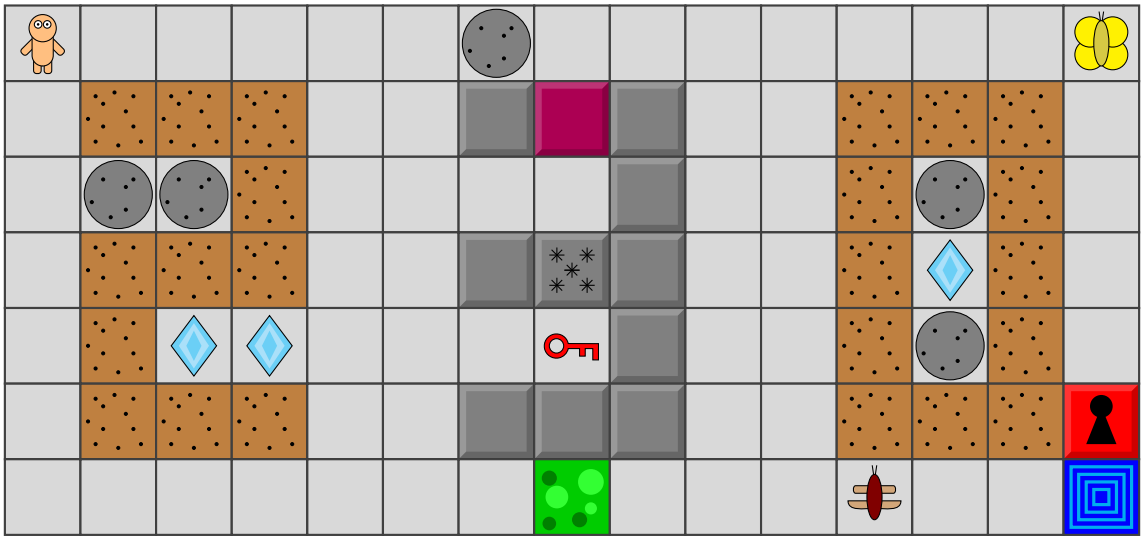
CS-230 Software Engineering
Functional Specification (2024/2025)

Liam O'Reilly



1 Introduction

You have been tasked with creating a digital version of the 1984 *Boulder Bash* game. There are many versions of this game from various ports to modern remakes. This is a game of strategy, quick response ... and maybe a little luck!

You can read a bit about Boulder Dash (optional) at `https://en.wikipedia.org/wiki/Boulder_Dash_(video_game)`. or watch a YouTube video at `https://www.youtube.com/watch?v=FiEVfa1OK_o`.

There is even an online version of the original available at `https://boulder-dash.com/online-free-game/`.

The game's rules are detailed in this document. They are not too complex, but not too simple either. Various design decisions have been taken to keep the complexity at a reasonable level and therefore may deviate from the original game. The rules specified here are a little loose. If they specify something then it must be followed, but otherwise you have creative freedom.

It is up to you to design the classes, algorithms and GUI involved in the development of this game.

The gameplay specified in this document **mostly** aligns with (a subset of) the original game, but does differ in places and functionality.

2 Game Title and Theme

You can come up with the title and theme of the game. You may stick with the theme of Boulder Dash or you may choose your own by putting some thought into this and making something unique and exciting!

This document describes the overall game idea, the gameplay and the rules. However, you may (and maybe should) substitute game elements (i.e., their names and graphics) to align with your own theme. For example, you could create a game around robots, dinosaurs, or whatever you like. The tiles and enemies (see Sections 3.1 and 3.2) may be substituted according to your theme as long as they behave as described in this document.

The graphics used in this document are fairly simple. You can and should produce graphics of equal or nicer quality if you are able to. Even simple image files, when tiled, can look very nice.

3 Overall Idea, Components, and Gameplay

The game comprises multiple levels (i.e., a collection of maps). A player plays one level at a time and, upon completion of a level, progresses onto the next level.

A level is made up of square tiles that form a 2-dimensional rectangular map in which the player and enemies travel. Your job as the player is to reach the exit. Each level is an underground cave where the vertical axis represents elevation (i.e., gravity pulls some things down).

Many of the details below leave certain quantities and durations open. The level files will specify these values (see Section 5).

The tiles that make up the level are all square and the same size. They are split into 3 categories: *Basic Tiles*, *Collectable Item Tiles*, and *Actors*.

3.1 Basic Tiles

The tiles that make up the level are all square and the same size. They are:

- Path:

  A Path is the most "normal" of floors. It can be walked on by the player and enemies.

- Dirt:

  Dirt can be walked on by the player. However, when the player walks on dirt, it is compacted and becomes a normal path. Enemies cannot travel over dirt.

- Normal walls:

  Normal walls block all movement. Neither the player nor the enemies can move over walls.

- Titanium walls:

  Like the normal wall, titanium walls block all movement. Neither the player nor the enemies can move over walls. Titanium walls cannot be destroyed by explosions (see Section 3.7).

- Magic walls:

  Like the normal wall, magic walls block all movement. Neither the player nor the enemies can move over walls. They interact with falling diamonds and boulders though (see Section 3.6).

- Exit:

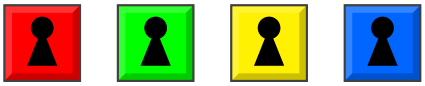
  The level is won if the player walks on an exit tile after collecting the required number of diamonds but before the level time elapses (see Section 3.10).

- Key:

  Keys come in four variations: *red*, *green*, *yellow*, or *blue*. Keys are collected when the player moves onto them.

- Locked Door:

  Locked doors come in four variations: *red*, *green*, *yellow*, and *blue*. A locked door acts as a wall, except in one situation. If a player attempts to move onto a locked door and has a key of the corresponding colour then the locked door opens and is replaced by a path.

  A key can only be used to open a single locked door (i.e., the key is consumed when opening a locked door).

3.2 Actors

The following tiles are actors (i.e., things which can move):

- Player:

  The player that is controlled using the keyboard.

- Boulder:

  Boulders block the path but they can be pushed (see Section 3.3. They also roll and fall (See Sections 3.4 and 3.5), and will kill enemies and the player (see Section 3.7).

- Diamond:

  Diamonds can be collected by the player by moving onto them. In fact this is required to win the

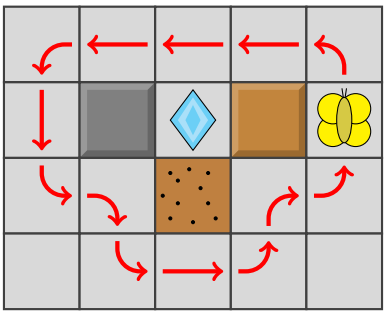
level (see Section 3.10). The also roll, fall, and kill like boulders (see Sections 3.4, 3.5, and 3.7).

- Butterfly (a type of enemy):

Butterflies can only travel over path tiles, and only if these tiles do not contain another enemy.

A butterfly tries to follow the edge of tiles it can travel on. This would be like following a maze keeping your hand on the wall to the left/right of you at all times. Each butterfly is initially set to either follow the left edge or the right edge when the level is loaded.

For example, a left-edge following butterfly would behave as follows:



Butterflies drop diamonds when destroyed by falling boulders and diamonds (see Section 3.7).

- Firefly (a type of enemy):

Fireflies behave the same as butterflies but do not drop diamonds when destroyed by falling boulders and diamonds (see Section 3.7).

- Frog (a type of monster):

Frogs can only travel over paths and only if these tiles to do not contain another enemy.

Frogs are the smartest of all the monsters and their mission is to seek out the player and kill them. They move rather slowly compared to the player. They move towards the player but do so by finding the shortest path through the level. If no path is possible for the frog to move then the frog moves in a random (but valid) direction. Note: This is one of the more difficult parts to implement.

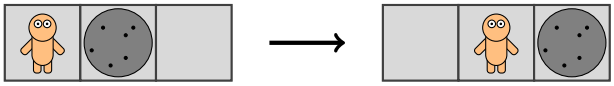
- Amoeba:

Amoeba is an organism that spreads (see Section 3.8).

Enemies refer to butterflies, and fireflies, and frogs. If an enemy finds itself directly next to the player in one of the four cardinal directions (directly above, directly below, directly left of, or directly right of) the player, the player is killed.

3.3 Pushing Boulders

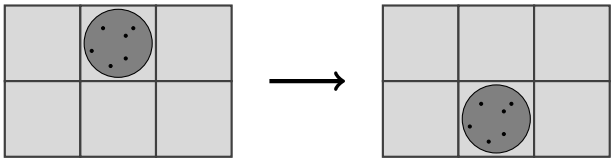
A boulder can be pushed by the player if, and only if, it is being pushed onto a path which is free of any enemies. For example:



Here, the player moves right and pushes the block onto an empty path tile.

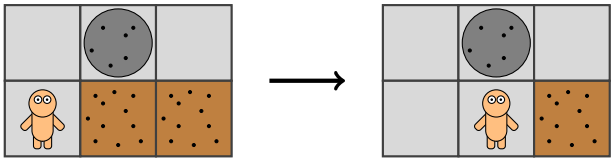
3.4 The Falling of Boulders and Diamonds

Both boulders and diamonds will fall if they are not supported by a tile underneath them. If the tile below a boulder or diamond is a path then it will fall. For example:



Note that the boulder could have rolled to the left or the right.

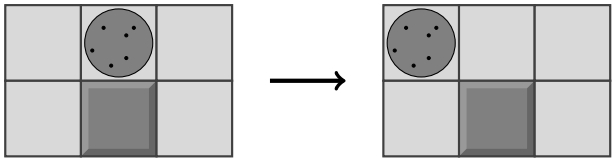
The player can move under a diamond or boulder without it falling. For example,



Here, the player is supporting the boulder. If the player moves to the left or right then the boulder will start to fall. If the player moves down then the boulder will also start to fall, but will kill them (see Section 3.7).

3.5 The rolling of Boulders and Diamonds

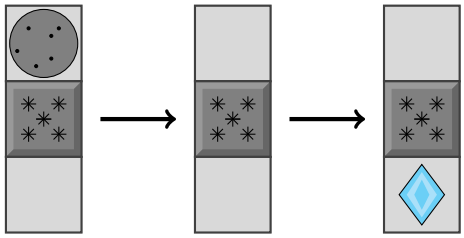
Boulders and diamonds can also roll sideways if space allows. If a boulder or diamond is on top of another curved tile (that is, a boulder, diamond, or wall) and the tile to the left or right, let's assume left, is an empty path with an empty path below it, then the boulder/-diamond will move to the left. It will then start to fall (see Section 3.4). For example:



Examples of situations where the boulder/diamond would not roll are:

3.6 Magic Walls

Magic walls allow boulders to be transformed into diamonds and diamonds to be transformed into boulders. Should a diamond fall into the top of a magic wall, it will fall through it and emerge as a boulder. Similarly, should a boulder fall into the top of a magic wall, then it will fall through it and emerge as a diamond. For example:

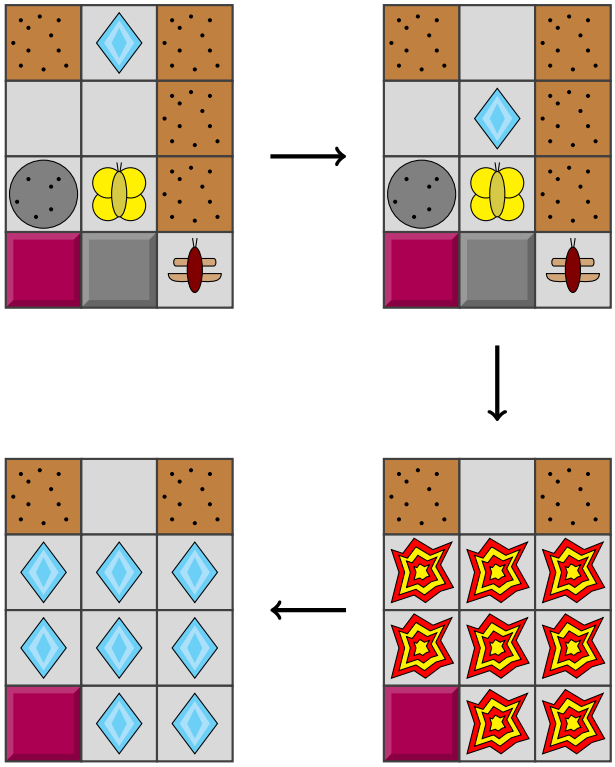


3.7 The Dangers of Falling Boulders and Diamonds

Falling boulders and diamonds can do serious damage.

If a falling boulder or diamond hits an enemy then the enemy explodes. Explosions happen in a 3x3 square with the hit enemy at the centre. The explosion lasts only briefly. Explosions affect tiles in different ways. Any dirt, normal walls, magic walls, locked doors, keys, boulders, diamonds, or amoebae caught up in an explosion are destroyed. Any enemy caught up in an explosion is also destroyed (but this does not generate a secondary explosion). Paths replace anything that is destroyed. Finally, if a butterfly is hit by a diamond or boulder, then after the 3x3 explosion dissipates diamonds are left in place of the explosions unless something could not be destroyed.

For example (assuming the enemies were not moving for simplicity):



If a falling boulder or diamond hits a player then the player dies. As a side note, in the original game, the player dies with an explosion that also drops diamonds, but this is only for aesthetics as the player is dead.

3.8 Amoeba and Its Spreading

Amoebae are harmless, except they spread. Each group of connected amoebae acts as one. Periodically, each group will spread to a random neighbouring cell (left, right, above or below) if possible. The rate of growth is specified in the level file. Amoebae can grow over empty path tiles and Dirt.

If a connected group of amoebae cannot grow, because growth is blocked, then the entire group turns to diamonds. If the connected group reaches a predefined size, specified in the level file, then they all turn to boulders (this acts as a sort of time delay which will probably block a large part of the level).

An amoeba kills any enemy it directly touches in one of the four cardinal directions (directly above, directly below, directly left of, or directly right of).

Finally, amoebae can be destroyed by explosions.

3.9 Game Ticks

This section describes one approach of achieving easy movement of actors. There would be other approaches, but this is the recommended approach.

At a regular interval a game tick occurs (say every 200ms). All movement happens via the game ticks. For example, the player might move on every 3rd game tick;

while the frog might move on every 5th game tick (hence is slower than the player); and the diamonds and boulders might roll and fall on every game tick (hence be quite fast). This allows different actors to behave/move at different speeds.

When the user presses a direction key, the game registers that direction and will attempt to move the player in that direction the next time the player moves in a game tick. Note: The act of pressing the key does not move the player (else, tapping the key quickly would cause the player to move faster than intended).

3.10 Winning and Losing the Level

In order to win the level, the player must reach the exit before the level time elapses and after collecting the required number of diamonds. The score is based on the number of collected diamonds and any remaining level time. The exact calculation is left open as a design choice.

You lose the level if you die or if the level time elapses.

4 Simplifications from the Original Game

The gameplay specified in this document mostly aligns to the original game, but does differ in places and functionality. Here is a non-exhaustive list of some of the more major aspects:

- Only a subset of tiles, items, and enemies are used in this document.
- While the game should be designed and implemented to have good-looking graphics, it does not need to have nice/fancy/smooth animations. Thus, a jumpy form of animation is fine. Of course, if you really want to you can implement a smooth style but this will be much much more difficult.
- Scrolling the board is not necessary (you can create the levels so that they fit in the window). That said, it would be a great extra feature. If you don't allow scrolling, make sure a large enough level can be accommodated so that it is interesting.

5 Levels

The game comprises of multiple levels. Upon completing a level the next level will be unlocked. Your player profile will keep track of the maximum level you have unlocked (see Section 6). You can replay all unlocked levels.

Each level is stored in its own file. It is up to you how you design and structure these files. However, they must be simple ASCII based files. This will make it easy for you to design multiple levels.

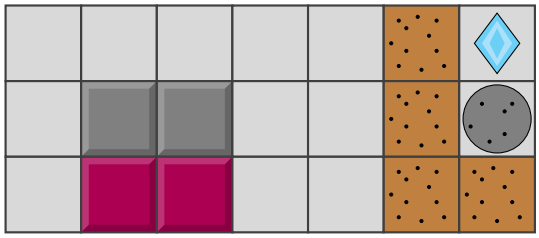
Each level must store (at least) the following information (not necessarily in this particular order):

- Size of the level (width and height) - the level need not be square.
- All the tiles and their locations.
- All related data of enemies, etc. For example,
  - For each butterfly and firefly, if it is left-edge following or right-edge following.
- The number of seconds that the player has to complete the level in (see Section 3.10).
- The number of diamonds needed to win (see Section 3.10).
- The rate that amoeba groups spread (see Section 3.8).
- The size limit for amoeba groups at which they transform into boulders (see Section 3.8).

A starting point of your design might be as follows:

```
1 7 3
2 PPPPPD*
3 PWWPPD@
4 PTTPPDD
```

The above specifies the tiles of the level shown below. The first two numbers specify the width and height of the level (which will make parsing the file easier). Following this, each row of the file represents one row of tiles of the level. With each tile being separated by a space.



Here, P represents a path, W represents a normal wall, T represents a titanium wall, D represents dirt, * represents a diamond, and @ represents a boulder. Note that not all tiles are shown in this example. Many tiles/enemies will need extra data to be specified.

Following this could come the rest of the data such as the time to complete the level, enemies, as well as all related data, etc.

There are many ways to design this. The important thing is to think about how you will parse the data. For example, by reading the width and height of the level first, it makes it much easier to read the tiles that follow

as you know how many will appear on each line of the file and also how many lines of tiles there will be).

6 Player Profiles

The application should support player profiles. Player profiles are used to record game stats for different players. Player profiles can be created and deleted via appropriate menu items or buttons within the application.

Each player profile:

- Has a player name.
- Keeps track of the maximum level that has been unlocked.

7 High Score Tables

The application keeps track of a high score table per level and also allows these to be displayed (per level). Each time a player completes a level, their profile name is recorded on the high score table for that level along with their score. Only the top 10 scores are kept (if a player scores less than all the top 10 for that level then they do not get added to the high score table).

8 Data Persistence

The player profile data is persisted across runs of the application. That is if the user quits the application, then upon reopening the application, the data is not lost.

9 Save/Load Games in Progress

The user shall have the ability to save an ongoing game. This will save the current game state to a file. The user can later load the saved game and resume play. All game data shall be in the same state as they were when the game was saved.

10 Extra Features

This section is not part of A1, but it will be part of A2. It will be helpful for you to plan for extensibility and hence know about this section.

To achieve top marks in A2, you will need to be creative. At a minimum, all the functionality of the Functional Specification should be completed to a high standard. All features should adhere strictly to the specification. You need to get all this working well in order to get a low First Class mark (in A2). In order to get higher marks, you are required to extend the implementation in novel ways. All extensions that do not violate the specification will be considered. Substantial extensions to the software, extra reading and learning, will be required to achieve a high First Class mark (in A2).

11 Forbidden Features

There are forbidden features which you must not design nor implement. These features are reserved for the assessment of CS-235.

Do not design nor implement any of the following features for CS-230:

- A level editor that allows users to create and design their own custom levels via a nice editor.
- Allowing multiple difficulties per level (where for example, the same level at a higher difficulty would have less time, etc.).

12 Libraries and Frameworks

You must program this game in Java using JavaFX. It is strongly recommended using JavaFX's Canvas class to draw the game.

You may use any classes and packages that are part of the standard Java SDK.

You may not use any other libraries or frameworks without first seeking approval. Please use the Canvas discussion board to ask such questions.

Game frameworks will **not** be allowed, this includes physics engines.