Junzhang Wang, jw4647
Bowen Zhang, bz896
Derek Huang, ch2699

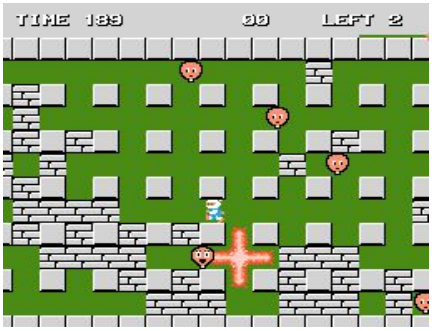
Professor Nathan Hull
CSCI-UA 480-002 iOS Programming
May 18th, 2019

Bubble War

Bubble War Documentation

Overview of the App

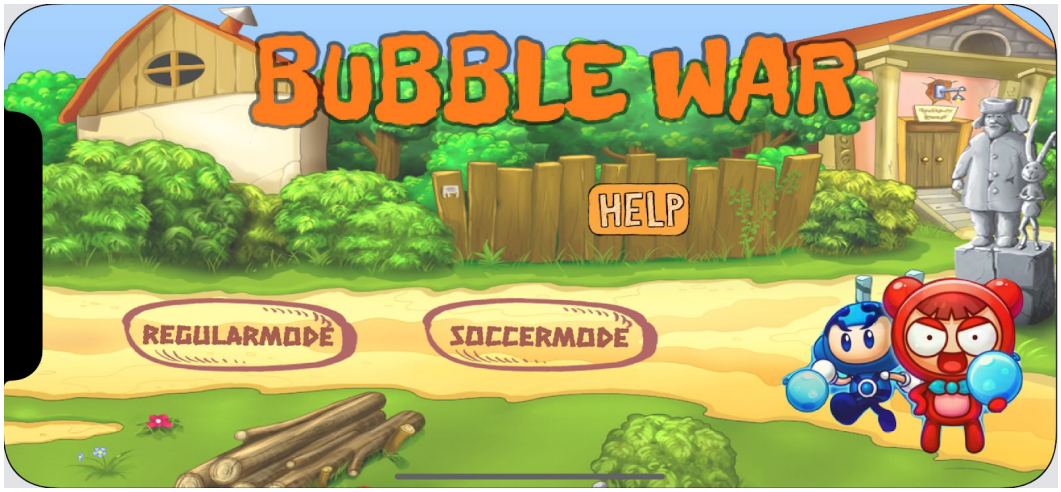
Background / Source of Idea



We made a maze-based iOS game with the idea inspired by the Japanese game Bomberman (link: https://en.wikipedia.org/wiki/Bomberman_(1983_video_game)) (see screenshot of the original game to the right). Our game is designed mainly to be a “User vs Robot” game, which means the users are able to play this game anywhere, without the need of connecting to Internet. The goal for the player is to collect power-ups by destroying blocks with bombs and ultimately defeat the robot by bomb explosion. The bomb explodes with a minor delay after being dropped, and the explosion extends to horizontal and vertical directions.

General Gameplay Setup

When the app loads, the main page will be presented to the user. We have designed two different game modes: Regular Mode and Soccer Mode (main_page). (The explanation of the game mode will come later in this document.) If the user choose “Regular Mode”, he then will have the option to select a map designed by us from the map library (game_selection_page). After that, the game will start and the user will be able to play the game (regularMode_gameScene). Once the game is finished, the user will be returned to the main page.



main_page

map_selection_page

regualrMode_gameScene

Map Elements

Each map will include elements such as destroyable blocks, non-destroyable blocks, flat lands (where players can move across), and various tools that may be used by the game characters to enhance their power. Under each destroyable block, there may or may not be a tool (generated randomly from the tool library designed by us). If there exists a tool under a destroyable block, after destroying that block using bubble, user can acquire that tool by simply walking to the cell that contains that tool, and then the effect of that tool will be applied to the user through various means such as change of power, temporarily acquiring special abilities and so on. Destroying any destroyable block will result in a flat land on the map.
Non-destroyable blocks, on the other hand do not generate any tools, and they will also block any explosion effect.

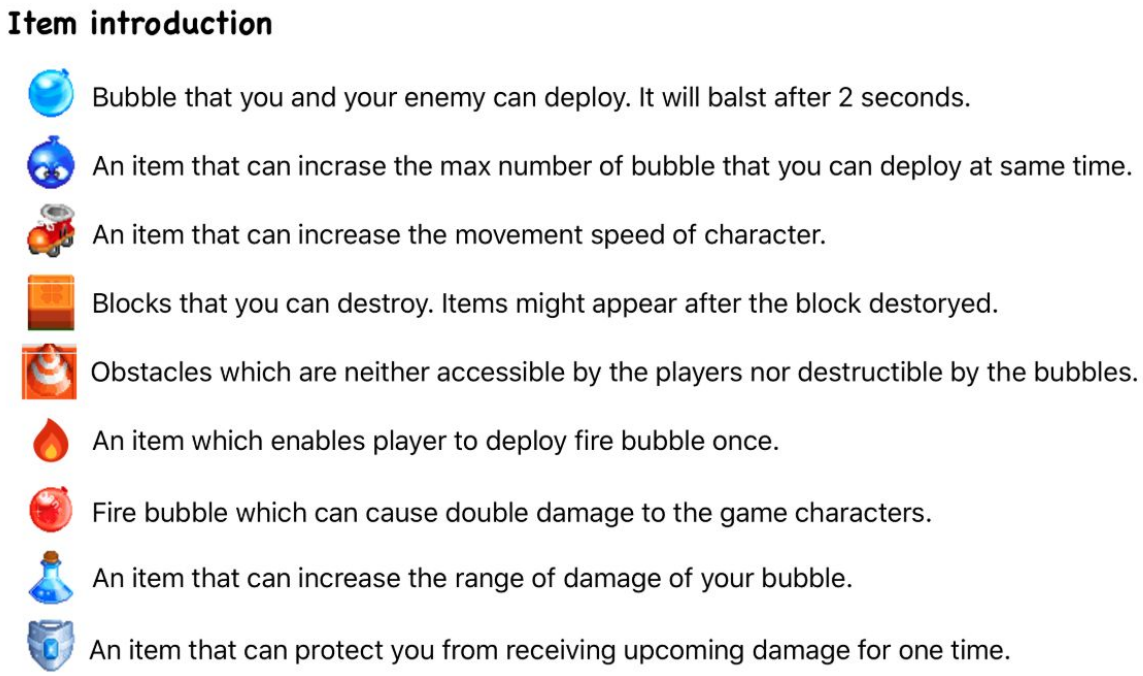
Basic Rules and Goals

The main task of each player (user or robots) is to defeat the opponent using bubbles they have. At the beginning of the game, the health of each player will be set to 3, the maximum bubble deploying capability for each player is one (i.e. the player can deploy only one bubble at a time and can not deploy the next bubble until the previously deployed bubble explode), and the range of damage of the bomb is one (the adjacent cells). A user's maximum bubble deploying capability and bubble damage will be strengthened by acquiring various tools (see the "Tools" section below for detailed explanation).

The explosion of the bomb can destroy the destroyable blocks on the map as well as reducing the affected opponents' lives by one. Once the opponent's health goes to 0, the user wins the round. The game time will be limited to 5 minutes. If no side wins when the game concludes, the game result will be a draw.

Tools

Different tools in the game have different effects that can strengthen player's particular ability.



Game Modes

There are two game modes implemented in the game.

The first game mode is called "Regular Mode". This game mode is simply the classic Bomberman game. The instructions are described above, and it is the main game mode that we suggest for the users.

The second game mode is called “Soccer Mode” (soccerMode_gameScene). We designed this to give a new twist to the original rules of the game. In Soccer Mode, instead of deploying bubbles, players will only be able to kick bubbles that are randomly generated on the map. Bubbles will be randomly generated on empty spots of map every 7 seconds. The number of bubbles generated will increase as time goes on. Each bubble generated will explode in 5 seconds and the power of these bubble will be fixed to 1. So, the players’ goal is to kick the bubble towards their opponent and let the bubbles’ explode next to the opponent. In order to kick a particular bubble, for instance, to the left, the player simply needs to approach the bubble from the right and keep walking left. The bubble will be kicked to 3 tiles away unless it hits another bubble, the other player, or an obstacle. In this mode, tools will also be generated randomly but with much lower possibility.

soccerMode_gameScene

Game Control

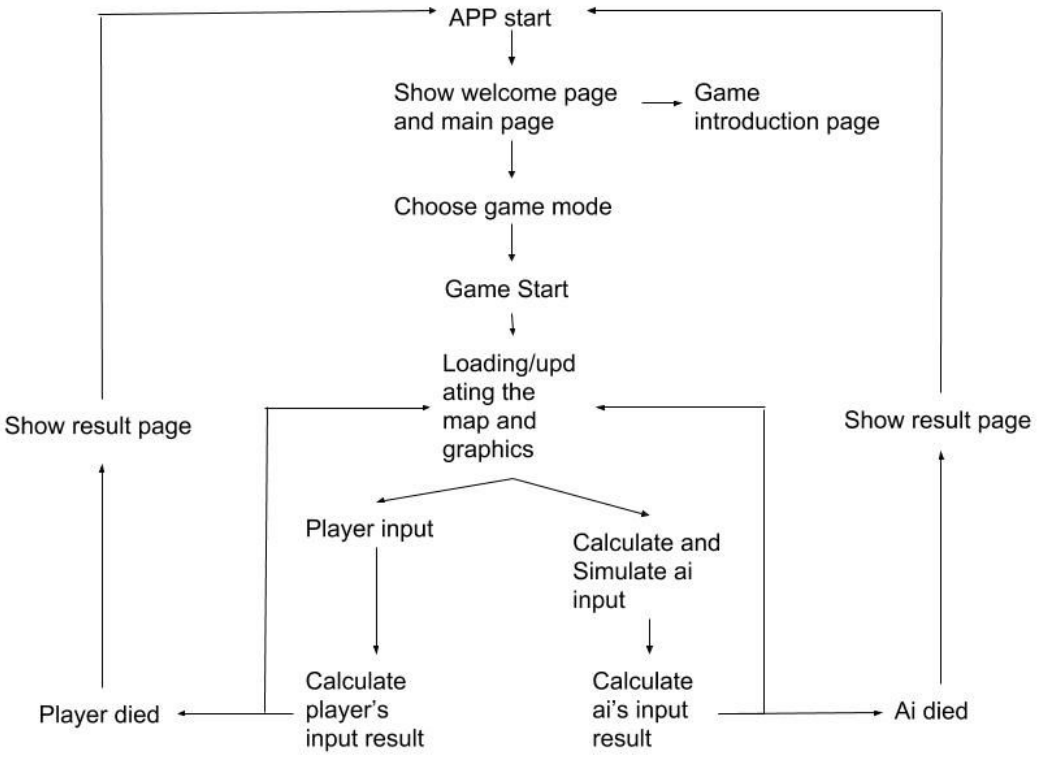
Intended Audience

People often have much short period idling time, such as on a subway, waiting for someone else, spare time before class, or when they simply get bored. This game's fast pace and exciting nature is a perfect choice for anyone who want to kill their idling time or simply want to have some fun.

Overall Logic and High-Level Flowchart

The game will start with the player choosing the game mode. Then, during the game, the program will take the gamer's actions, such as placing the bubble, moving or using items, continuously. At the same time, the program will simulate the "robot's" movements. After this, the program will calculate the result of such actions and reflect the result by updating the game board (or character status). The game will end when either of the gamer's or robot's health goes to zero or the time is up.

Below is the detailed High-level Flowchart.



Technical Description

There are several technical challenges that we have conquered.

(1) Initialization of the game map:

We used SKScene for creating the map. In the SKScene file, we made use of the SK Tile Map Node to build our block-based map. The advantage of using this technique is that the SK Tile Map Node contains all the elements that we will need to create the map. By putting map tiles on this node, we can easily manage the location of each element. We don't have to create each element as an individual SKNode and manually put them together to create the map. It can also help us create the test scenarios easily (ie. We can quickly add block or bubble around the Robot to create test scenarios for logic testing). In addition to this, SK Tile Map Node can quickly make conversion between row & column of the tilemap and the position by pixel on the screen (ie. We convert the player's location to row and column of the tilemap to check the object around it.).

(2) Players' movement and abilities:

A simple way of achieving player displacement is to relocate the player and re-render the player at the new tile where it should be showed. However, there would be a sense of discontinuity for the user if such method is implemented. In order to allow a smoother gameplay experience, we adopted the method of movement by pixels. We added a velocity for the character whenever the move button is pressed. In this case, the character would be rendered 60 times per second and each time with a minor displacement.

(3) Movement autocorrection

Since we are implementing pixel movements on top of a matrix map, minor issues occur such as player character will stand on the edge of tiles, causing half of its body layering with other objects, or sometimes failure to continue moving in certain directions. To solve these issues, we developed a multi-point detection algorithm to autocorrect the character's position.

In simple terms, the multi-point detection algorithm imagines there are "satellites" around the character, and pushes it back to the center of the tile.

The code can be found under the shiftDodgeDeployedBubble() function within the Robot.swift file.

(4) Robot Algorithm:

The entire robot algorithm can be found in the Robot.swift file.

We have designed a challenging robot to compete with the user using the following algorithmic structure:

When the player is over 70 pixels away from the robot, the robot will check for surrounding blocks and destroy them to collect power-ups, the robot will move toward user player if there is nothing found. When the user player is within a 70-pixel distance, the robot will start chasing the user player and deploying bubbles around the user player.

Under this broad structure, there are many detailed algorithms implemented, such as dropping and dodging bubbles.

1. All possible routes within a certain range will be checked by the robot to see if it is able to avoid being bombed if the robot drops a bubble before actually deploying the bubble.
2. All possible routes within a certain range will be checked to dodge a surrounding bubble
3. While dodging diagonally is prioritized, dodging vertically or horizontally is also implemented, since the robot knows its own power of bubble explosion.

4. Dodging two consecutive bubbles is also implemented. This is especially useful when the robot or user player acquired more than one bubble per term, or when the two characters are very close to each other.

(5) Graphic design, in-game animations and sounds

Background music is played when the app is loaded. Different music will be played according to different view controllers that the user is seeing. During gameplay, every time a bubble explodes, a player receives damage, or when a player is moving around, there are in-game animations and sound effects that indicate the trigger of these events. This is also an important feature that enhances user gameplay experience.

(6) Controller System

First of all, we have created a SKSpriteNode called Controller (Controller.swift) with clear texture, size of the entire screen, and user interactions is enabled. On the Controller Node, we add all the buttons we needed for user operations and override all the touches methods needed to handle finger movements on the screen.

Then we have a Controller input handler protocol (ControlInputHandler.swift) which is extended by a GKComponent called Controller component (ControllerComponent.swift), which then is added as an instance of the controller node and also linked to the player entity in the GameScene.

After implementing the controller input handler protocol, this controller component can now handle all messages received from the controller node. Here we have another two protocols called move delegate and deploy bubble delegate which are extended by the GameScenes. And then we add the GameScenes as both the move delegate and the deploy bubble delegate of the controller component. This is how we execute the messages received from the controller.

Teamwork/Responsibilities

We divided this project into several tasks and assigned a set of tasks for each person to work on individually. Besides individual tasks, we also meet regularly to check status, exchange our findings and solve the problems we encounter during the project development.

- Game Logic
    - Controller system (user movements, user actions (dropping bubbles, using tools) **- Junzhang**
    - Classic mode / Soccer mode settings **- Bowen**
    - Implementing functions of tools **- Bowen/Junzhang**
    - Random item generation **- Derek**
- Robot Logic

    - A clever enough robot player that allows an enjoyable game play experience for the user **- Derek/Junzhang/Bowen**
- In-Game Design
    - Creating maps with destructible blocks, trees, non-destructible rocks, and (Map Library Design) **- Junzhang/Bowen**
    - Real time status bar(bubble power, health, etc) **- Junzhang**
    - Sound effects and background music **- Bowen**
    - In-game animation (Event triggered, Player move actions, Rendering items etc.) **- Bowen**
- User Interface Design
    - Main Page, Instruction Page(Connecting Different Views) **- Bowen**

Features Not Implemented & Future Enhancements

In the future, we would like our games to have following added features:

- Allow online/bluetooth battle between players.
- Allow player to see their match statistics, such as number of games won.
- Add fancier map/tools/power-ups such as traps/ice.
- Add more game modes.
- Improve robot algorithm and allow different level of difficulties.