Zusammenfassung der Arbeit Abstract of Thesis	
Fachbereich : Department	**Electrical Engineering and Computer Science**
Studiengang : University course	**Information Technology**
Thema : Subject :	**Rebuild of a flash game using JavaScript**
Zusammenfassung : Abstract :	

The capability of JavaScript in terms of game programming is often underestimated. This thesis is aiming at rebuilding a flash game using JavaScript. The thesis paper includes the details of the primary processes of software development phases, while the derivable is a standalone browser game. Not only does it show the capability of JavaScript, but also demonstrates the process of adopting a scientific software development approach.

Verfasser : Author :	**QIU Zhenzhou**
Betreuender Professor/i Attending Professor :	**Prof. Dr. Nane Kratzke**
WS / SS :	**SS<2014>**

Fachbereich Elektrotechnik und Informatik
Dekanat - Prüfungsausschuss

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

Declaration of the Candidate

I, the undersigned, hereby declare that the work contained in this thesis is my own original work, and has not previously in its entirety or in part been submitted at any university for a degree.

Only the sources cited in the document have been used in this draft. Parts that are direct quotes or paraphrases are identified as such.

I agree that my work is published, in particular that the work is presented to third parties
for inspection or copies of the work can be passed on to third parties.

Date	Signature

Table of Contents

1 Introduction

1.1 Motivation

Technical capabilities of JavaScript are often underestimated. The traditional perspective believed that JavaScript is not a good choice for game development. The aim of this thesis is to analyze the Bomberman and derive a more interesting variation, and demonstrate technical capabilities of JavaScript by implementing the variation. The variation game would be named Popoman.

1.2 Scope

- Popoman should be purely developed in JavaScript.
- Popoman should be a standalone browser game.
- Popoman should adopt an MVC-based architecture.
- Popoman should be tested and evaluated in certain testing strategy.
- Popoman should be able to run properly on Chrome, Safari and Firefox those mainstream browsers.

1.3 Organization

The rest of the document is structured as followed. Chapter 2 is Background. Chapter 3 is Requirement Analysis. Chapter 4 is System Architecture Design. Chapter 5 is Functional Detail Design. Chapter 6 is Implementation Description. Chapter 7 is Test and Evaluation. Chapter 8 is Conclusion And Outlook. In the references lists the referent links and materials.

2 Background

2.1 Bomberman

Bomberman (also known as Dynablaster or Dyna Blaster in Europe) is a strategic, maze-based video game franchise originally developed by Hudson Soft. The general goal throughout the series is to complete the levels by strategically placing bombs in order to kill enemies and destroy obstacles. Exploding bombs can set off other bombs, kill or injure enemies, and destroy obstacles. However, they can also kill or injure the player character, destroy props, and sometimes "anger" the exit, causing it to generate more enemies. Most Bomberman games also feature a multiplayer mode, where other Bombermen act as opponents, and the last one standing is the winner. In this mode, props are plentiful [1].

2.2 JavaScript perspective in game development

JavaScript (JS) is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed. It is also being used in server-side network programming (with Node.js), game development and the creation of desktop and mobile applications [2].

Even JavaScript is regard to be powerful in terms of Internet programming, however, no one can deny the fact that the primary usage of JavaScript is focusing on website development, especially on the client-side, in charge of handling interaction between user and browser.

JavaScript is not convinced to have a promising future in terms of game development by traditional view. The uppermost reason is that they claim JavaScript is not an object-oriented language. JavaScript is a featherweight script language, compared with programming language C#, C++ and Java, however, it is still a real object-oriented programming language. JavaScript does not have classes, but it does have constructers that do what classes do, including acting as containers for class variables and methods. It does not have class-oriented inheritance, but it does have prototype-oriented inheritance. Some argue that JavaScript is not truly object oriented because it does not provide information hiding. That is, objects cannot have private variables and private methods: All members are public. But it turns out that JavaScript objects can have private variables and private methods [3]. All these indicate JavaScript is object-oriented, capable of object-oriented programming which is essential for game development. Despite object-oriented, another reason people refuse to develop game in JavaScript is that they think the DOM element is difficult to manipulate, and unsuitable for game that contains many objects. Indeed, manipulating a large amount of DOM elements on the browser can result in severe performance problem. Such shortcoming really limits the capability for JavaScript to develop a complex game. Fortunately, for those games with little objects involved, DOM is still worth considering. With HTML5 gaining more and more popularity nowadays, most of the mainstream browsers have support HTML5 features in their latest versions. Meanwhile, HTML5 Canvas has become a new symbol for JavaScript game development. Hundreds of HTML5 Canvas game engines arose overnight. The Canvas element is a new element in HTML5 that allows developers to draw graphs, graphics, games, art, and other visuals right on the web page in real time in the user's browser [4]. In conclusion, the developer is given an exciting new approach to develop game with JavaScript.

2.3 Canvas VS DOM

There are primarily two popular approaches to develop game in JavaScript. HTML5 Canvas and DOM.

The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. Objects in the DOM tree may be addressed and manipulated by using methods on the objects. The public interface of a DOM is specified in its Application Programming Interface (API). [5] DOM allows users to dynamically manipulating the display of the page content, such as dynamically show or hide an element, change their properties and so on.

Canvas is an element of HTML5 and allows for dynamic, scriptable rendering of 2D shapes and bitmap images. Canvas consists of a drawable region defined in HTML code with height and width attributes. JavaScript code may access the area through a full set of drawing functions similar to those of other common 2D APIs, thus allowing for dynamically generated graphics. Some anticipated uses of canvas include building graphs, animations, games, and image composition [6].

In the field of JavaScript game development, DOM and HTML5 Canvas are often compared with each other. Canvas is highly appreciated for its capability of free drawing feature. It is able to manipulate pixel with the help of Canvas, which allows users to create fascinating animations. DOM does not have such powerful capability of pixel drawing, but it overweighs Canvas in its well-structured element manipulation feature. Each element on the browser that created in DOM is manageable, which is convenient to establish a highly user interacted game. Furthermore, when it comes to compatibility, game developed by DOM can support more browsers, while HTML5 Canvas cannot. Because HTML5 is a new standard, some of obsoleted browsers don’t support it, such IE 6,7,8,9.

Popoman is to be implemented by DOM for following reasons.

1. In order to achieve a preferable compatibility among different browsers.
2. Although Canvas provide powerful drawing feature, it is not so useful in our case since most of the game images resources collected are in GIF format.

3 Requirement Analysis

3.1 Analysis and refinement

Due to various versions of Bomberman available, a specific online version is selected to be the prototype. This flash Bomberman version is available online.

(http://www.7k7k.com/swf/396.htm)

Analysis is conducted on THREE steps.

1. Game concept introduction. Aiming at understanding the game concept, introducing the game related terms, and clarifying the interactions between various game objects.

2. Game concept refinement. Aiming at analyzing the game concept, try to derive a more interesting game concept by making it more sophisticated.

3. Player interaction evaluation. Aiming at analyzing the game in terms of player interaction, evaluating and improving the handling.

3.1.1 Game concept introduction

In this online Bomberman version, it contains two modes – PvP (Player vs Player) and PvC (Player vs Computer). In the screenshot there are two roles (Fig 3-1). In PvP mode, player-1 controls the white role, while player-2 controls the yellow role. In PvC mode, player-1 controls the white role, while the yellow role is controlled by game-AI. Bomberman is a grid-based game. A new battle arena is created by game when a new round start, there are some baffles being placed in grids between two roles. Role can release a bomb on his position. Bomb with a timer that will explode within a short time, popping and releasing flame both vertically and horizontally. Crisp baffles on the arena can be destroyed by flame, while firm baffles are indestructible. Roles who are exposed to flame will die. The objective of this game is to strategically place bomb to kill the opponent. The playtime for each round is limited.

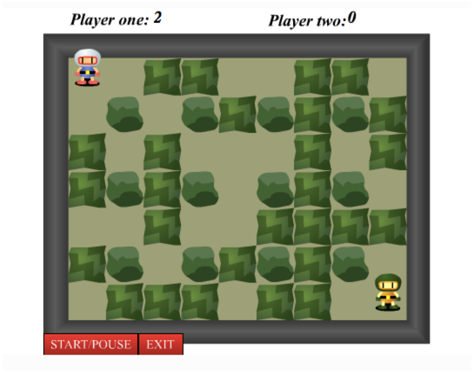


Fig 3-1

Terms & Definition

arena: Playground where the battle took place.

grid: Cell on arena.

role: Small figure control either by player or game-AI.

baffle: Surrounding stationary obstacle that placed on arena between roles.

crisp baffle: A type of baffle can be destroyed by flame.

firm baffle: A type of baffle cannot be destroyed by flame.

Fig 3-2 shows the Game Object Tree of Bomberman.

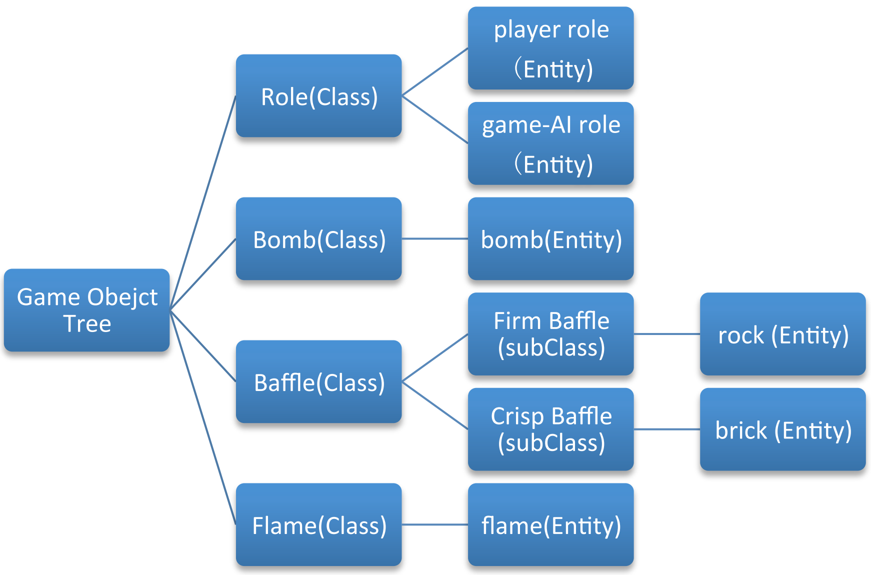


Fig 3-2 Game Object Tree of Bomberman

3.1.2 Game concept refinement

Game concept refinement is conducted in two perspectives.

3.1.2.1 Functional perspective

Drawback:

1. Role is only able to carry one bomb with him, significantly restrict his capability to pose threat to opponents.
2. Bomb power is fixed to Level-1, which is not powerful and flexible enough.
3. Role speed is fixed, not having different levels.

Refinement:

The proposed solution is to add a new type of game object Prop to Game Object Tree. Props are the bonus items hidden inside baffles on arena. They appear when baffles are destroyed by flame. Here we define THREE props.

Bomb-up: increase the bomb amount of role by one.
Power-up: increase the bomb power level of role by one grid.
Speed-up: increase the walking speed level of role by one.

Therefore, role is able to carry more bombs with him, with a limited maximum value. Bomb power is varied from Level-1 to Level-Max. It is possible to cover more grids than before. Role speed is varied from Level-1 to Level-Max. Different roles can have different speeds.

With the possibility of more bombs appearing on arena, it is essential to derive a bomb explosion mechanism to handling the bombs on the arena. It would be more rational to allow the flame to detonate bombs on the arena.

Expected Result:

1. Derive a new type of game object Prop.
2. Derive a bomb explosion mechanism, which will handle the explosion range.

3.1.2.2 Non-functional Perspective

Drawback:

1. The image style of Bomberman is a little old-fashioned.
2. The images of baffles are too monotonous, not colorful enough.

Refinement:

1. Replace bomberman figures with other characters.
2. Replace bombs with water balloons.
3. Replace flames with water sprays.
4. Replace original monotonous baffles with more various and colorful baffles.

Result:

Image style of Popoman is changed to be more appealing, different from Bomberman. The images can be found in 3.3.6 Game Object Prototyping.

Fig 3-3 shows the Game Object Tree of Popoman.

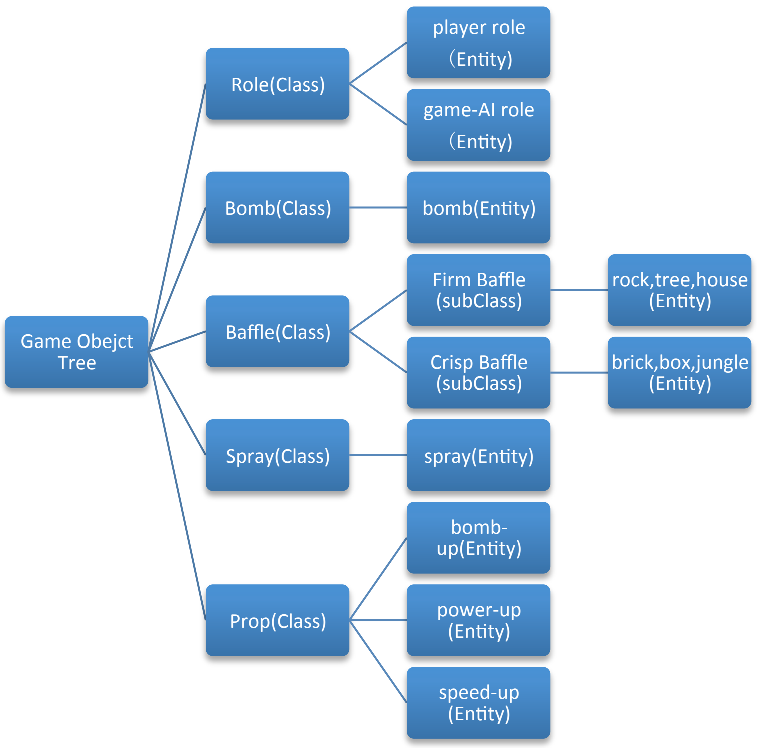


Fig 3-3 Game Object Tree of Popoman

3.1.3 Player interaction evaluation

In Bomberman, player-control is via keyboard. Using Up, Down, Left and Right these four arrow keys to move the role in four directions, and the Space is placing the bomb.

Drawback:

1. Jumping-style movement. The biggest drawback in Bomberman in terms of player interaction is its Jumping-style movement. The role movement is grid-based in Bomberman. When player presses a certain arrow key demand the role to move, the role will not move smoothly in desired direction but directly jump to another grid. Such jumping-style movement is awkward because of lacking of smoothness and successiveness.
2. Key-state tracking. In Bomberman, there is an obvious phenomenon when controlling the role to move around. If player firstly pressed the Left, role moved to the left. Then without releasing the Left, player pressed the Right, this time the role moved to the right. This reaction is reasonable, since the Right was pressed later. The role should response to the new demand. But then the awkward things happened, if player released the Right, the role just stopped moving. Notice that the Left was still pressed by player. In this case, it is unreasonable for the role not to move to the left again. Such disappointing result is due to a lacking of key-state tracking mechanism. The game

needs a mechanism to keep tracking of the state of each direction key – whether they are under pressed.

Refinement:

1. Define a new approach of role movement, not based on grids. In this new approach, role doesn’t jump from one grid to another, but is able to move freely on arena like in other 2D RPG games.
2. Due to a more loose-style movement, collisions between roles and other stationary objects inevitably happen. Therefore, a collision detection-handling mechanism should be derived in Popoman, aiming at achieving a rational movement of role and avoiding improper overlap of images.
3. A key-state tracking mechanism should be realized in Popoman, aiming at providing a more reasonable and comfortable player interaction in terms of direction controlling of role.

Expected Result:

1. A new role movement approach.
2. A collision detection-handling mechanism.
3. A key-state tracking mechanism.

3.2 Requirement Specification of Popoman

3.2.1 Functional Requirements

Functional requirements define what a system is supposed to accomplish.

Role Movement Requirement (RM)

RM.RS001	Role can move freely, stop at any position, without restriction of grid.
RM.RS002	Role cannot move outside the arena.
RM.RS003	Role cannot walk through brick, house, box and tree.
RM.RS004	Role can walk through jungle and role.
RM.RS005	Role cannot walk through bomb.
RM.RS006	Collision can be handled.
RM.RS007	A stable key state tracking mechanism is required.

Bomb Event Requirement (B)

B.RS001	Role can release a bomb on an empty grid when he has remaining bombs.
B.RS002	Role cannot release a bomb when he does not have any remaining bomb.
B.RS003	Bomb cannot be released on a grid that already contained a bomb.
B.RS004	Released bomb can explode after certain time, popping out cross-shaped spray on arena.
B.RS005	Spray cannot propagate through the border.
B.RS006	Spray cannot propagate through the tree, house, box, and brick.
B.RS007	Spray can propagate through jungle and role.
B.RS008	Bomb power level should be equal to the power level of role.
B.RS009	Bomb within another bomb's explosion range should be detonated, and they should explode simultaneously.
B.RS010	Spray can destroy role.
B.RS011	Spray cannot destroy tree and house.
B.RS012	Spray cannot destroy box, brick and jungle.

Prop Event (P)

P.RS001	Prop can be generated from a destroyed box, brick or jungle.
P.RS002	Bomb-up can be collected by role. If the bomb of role isn't max, bomb amount plus 1.
P.RS003	Power-up can be collected by role. If the bomb power of role isn't max, bomb level plus 1.
P.RS004	Speed-up can be collected by role. If the speed of role isn't max, speed level plus 1.

Sound (S)

S.RS001	A BGM should be played and looped. And the BGM should not begin again when screen switching happens.
S.RS002	A sound should be played when a new round begin.
S.RS003	A sound should be played when bomb released.
S.RS004	A sound should be played when bomb explode.
S.RS005	A sound should be played when prop collected.
S.RS006	A sound should be played when box pushed.
S.RS007	A sound should be played when role killed.
S.RS008	A BGM should be played and looped during the whole round.
S.RS009	A sound should be played when the result is WIN.
S.RS010	A sound should be played when the result is LOSE.
S.RS011	A sound should be played when the result is DRAW.

Game Logic (GL)

GL.RS001	In solo mode, kill the game-AI opponent within the playtime will result WIN.
GL.RS002	In solo mode, being killed within the playtime will result LOSE.
GL.RS003	In multi mode, if main-role kills sub-role, result is 'Main Role WIN'
GL.RS004	In multi mode, if sub-role kills main-role, result is 'Sub Role WIN'
GL.RS005	In multi mode, if main-role and sub-role are alive until the playtime run out, result is 'DRAW'

Menu UI (UI)

UI.RS001	A splash screen that can navigate to mode screen.

UI.RS002	A mode screen where player can choose play mode, and can navigate to map screen.
UI.RS003	A map screen where player can choose play map, and can navigate to game screen beginning the new round.
UI.RS004	A game screen that contain the arena.
UI.RS005	A result screen that shows the result when the round is over.

Game-AI (AI)

G.RS001	Game-AI role can make proper decision based on the environment, including dodging obstacles and bombs.

3.2.2 Non-functional Requirements

Non-functional requirements specify criteria that can be used to judge the operation of a system.

NF.RS001	Compatibility. The game should be playable on latest version of mainstream browsers – Chrome, Safari and Firefox.
NF.RS002	Security. The game source code should be protected in certain method, preventing from easily access.

3.3 Prototyping

3.3.1 Splash Screen

The game begins with the splash screen. It has a NEW button linked to mode screen.

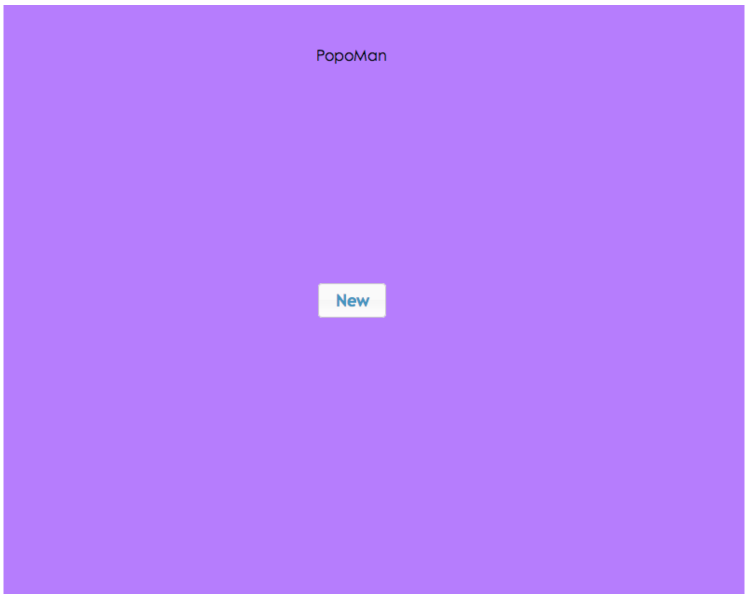


Fig 3-4

3.3.2 Mode Screen

Mode screen has two buttons SOLO and MULTI for player to select the desired mode. Both of them are navigated to map screen.

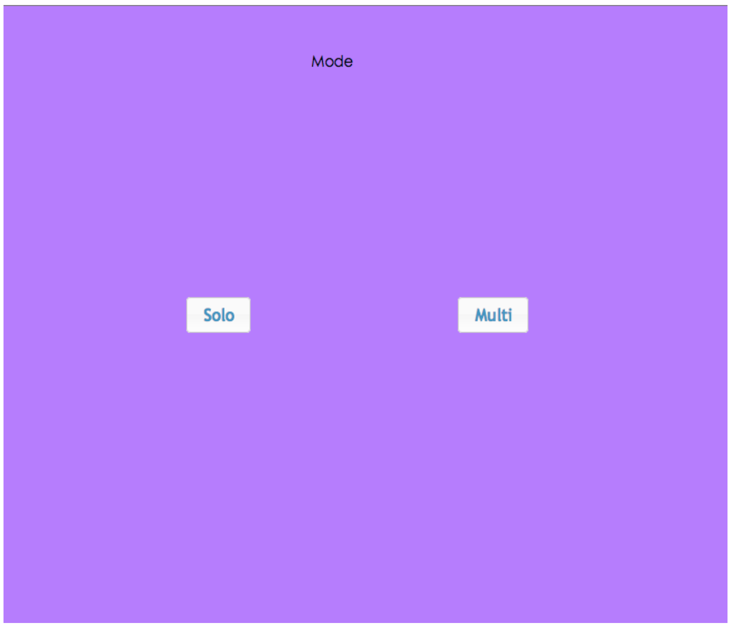


Fig 3-5

3.3.3 Map Screen

Map screen has a pull-down menu for player to select desired maps, as well as a GO button for player to begin the new round.

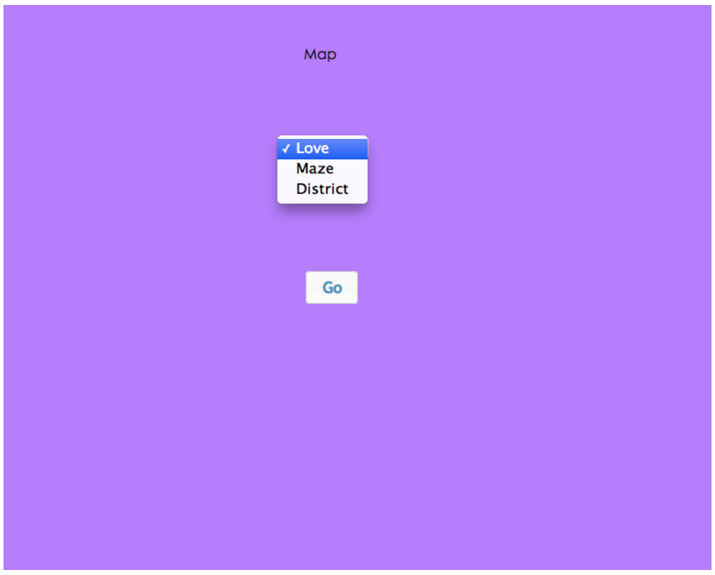


Fig 3-6

3.3.4 Game screen

Game screen contains the battle arena and all the game objects. When the round is over it will be redirected to result screen automatically.

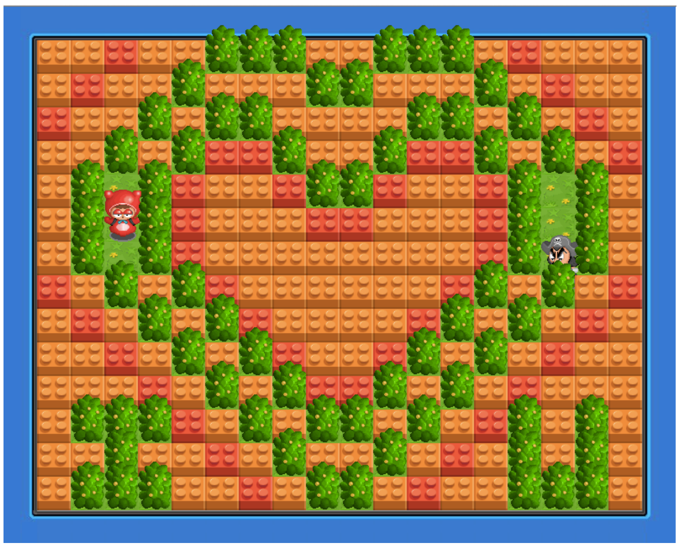


Fig 3-7

3.3.5 Result screen

Result screen is responsible for showing the result. It also has a BACK button to back to splash screen.

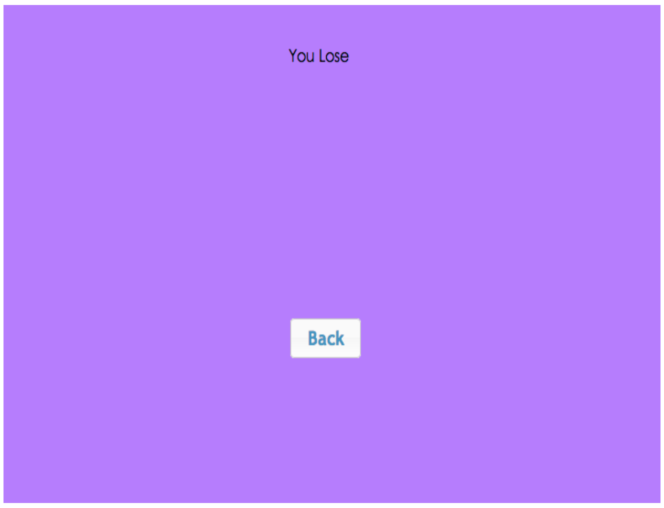
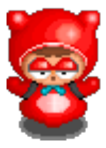
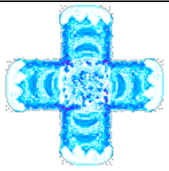


Fig 3-8

3.3.6 Game Object Prototyping

Object	Description	Prototype image
Player role	Role controlled by player via keyboard.	
Game-AI role	Role controlled by game-AI.	

Brick	Baffle that can be destroyed by spray.	
Box	Baffle that can be destroyed by spray.	
Jungle	Baffle that can be destroyed by spray.	
Tree	Baffle that cannot be destroyed by spray.	
House	Baffle that cannot be destroyed by spray.	
Bomb	Bomb that released by role. It can pop cross-shaped spray when exploding.	
Spray	Cross-shaped water created by exploding bomb.	
Bomb-up	Prop that can increase the bomb number of role	
Power-up	Prop that can increase the bomb power level of role.	
Speed-up	Prop that can increase the walking speed level of role.	

4 System Architecture Design

The architecture of Popoman primarily contains two parts. System part and game object part. System part is in charge of the main logic of game, including screen switching, mode selection, map selection, arena establishment and result judgment. Game object part is focusing on defining suitable classes for all game objects in Game Object Tree of Popoman.

4.1 System part

Game class is defined to handle the main logic of game. It contains methods correlated with screen switching, arena establishment and result judgment. Some classes are defined to separate the task of Game, aiming at avoiding a giant Game class. As shown in Fig 4-1, responsibility of each class is indicated by name.

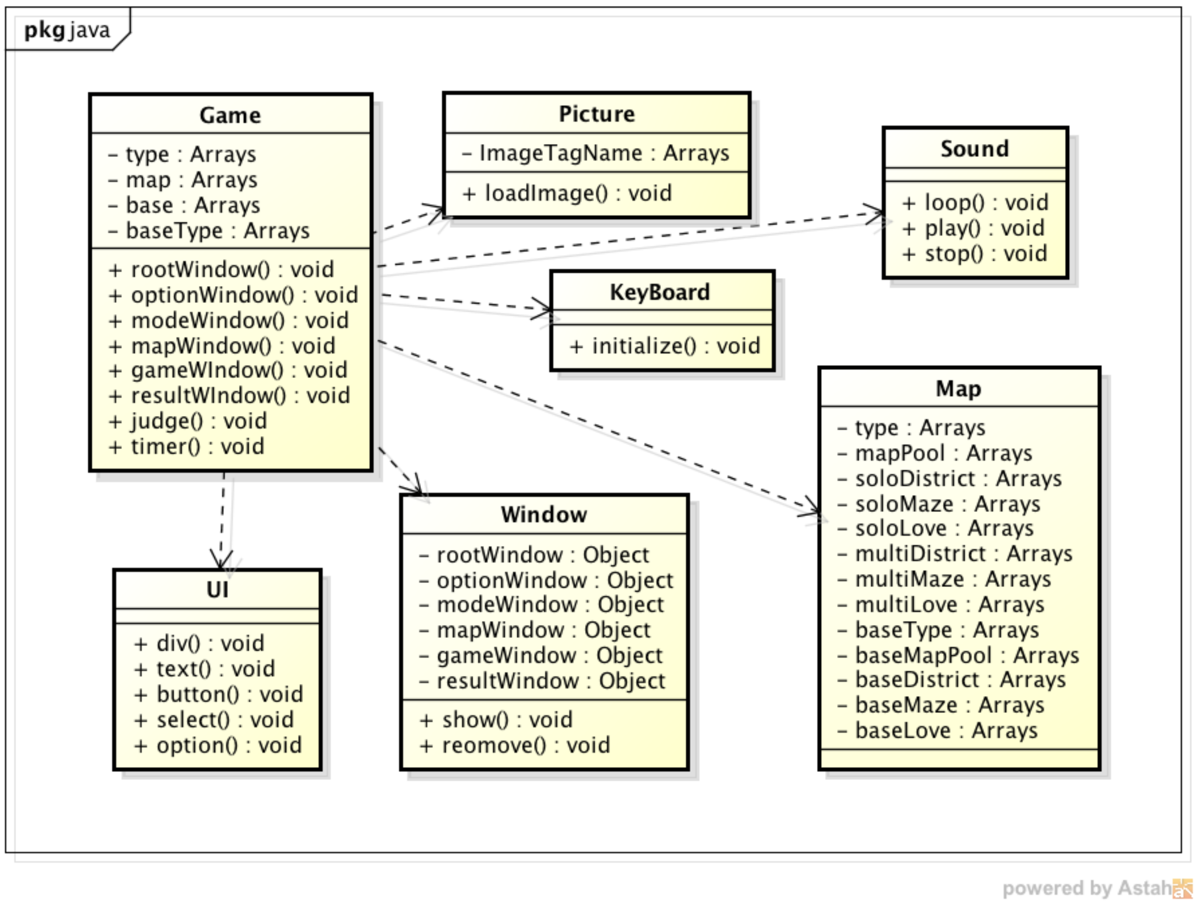


Fig 4-1

Here are the brief functional descriptions of each class.

Window: It contains all the screen elements, with methods to display and remove screens.

UI: It has methods to create the components on the screen, such as a text or button.

Picture: It is in charge of preloading images as the game initializing.

Sound: It has methods to control the sound in the game.

Map: It contains all the maps of the game, which are stored in the form of two-dimensional arrays.

Keyboard: It contains all the information of key in terms of setting and state.

4.2 Game object part

The classes are defined based on the Game Object Tree of Popoman (Fig 3-3). In view of similar behaviors between player role and game-AI role, it is suitable to define a Role Class to handle both of them. As for baffle, a Baffle Class is defined and the destructibility is considered as an internal attribute. Fig 4-2 demonstrates the relationship between classes and entities.

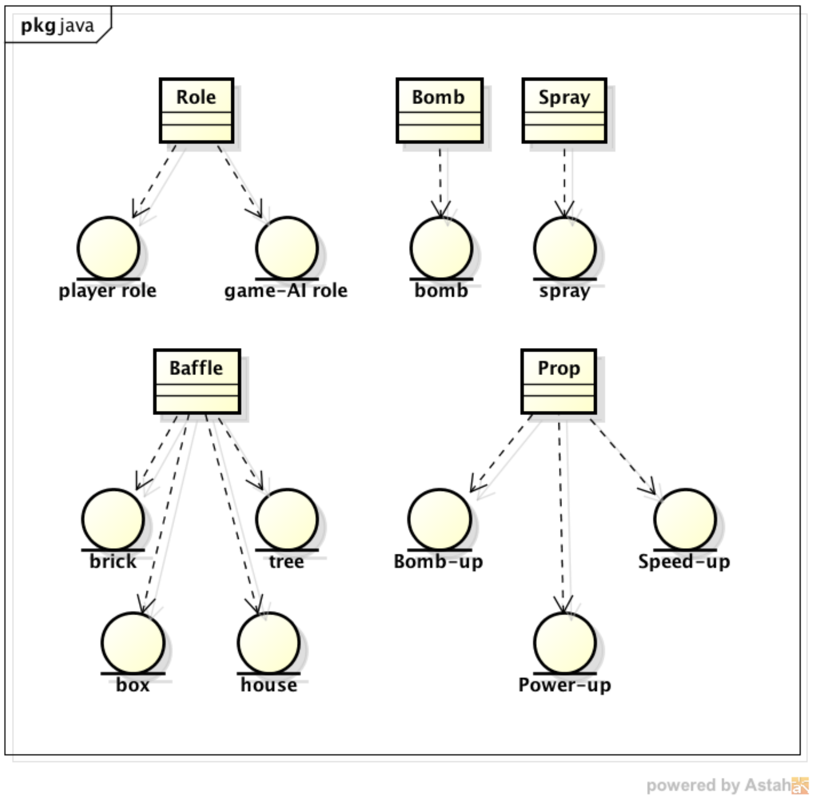


Fig 4-2

The game is originally attempt to adopting the architecture above. However, during the prototyping implementation, a problem arose. For example, because of using single Role class to handle the role, it should contain all the associated methods from attribute modification to image presentation, as well as the event handling. Such giant Role class is hard to implement and maintain. It would be more useful if there exists an approach to separate the tasks inside one class. Therefore MVC pattern is proposed to fulfill the task. Here is a short introduction of MVC.

In object-oriented programming development, model-view-controller (MVC) is the name of a methodology or design pattern for successfully and efficiently approach to separate the underlying data models with the presentation.

Model is where the underlying data manipulation took place. It has methods to retrieve, modify and update the data. Model can communicate with controller only.

View is where the presentation of data took place. In JavaScript, the approach to modify the presentation of elements on the browser is to alter the CSS attributes. Therefore, view has methods correlated with CSS modification. View can communicate with control only.

Controller is the bridge between model and view. It is in charge of selecting appropriate model and corresponding view.

From the above description of MVC, obviously it is an ideal approach to separate attribute modification, image presentation, as well as the event handling into three parts. Therefore the idea to apply MVC pattern in object layer is putting forward. Although this structure might be regarded as overdesigned from current point of view, which will be discussed in later section.

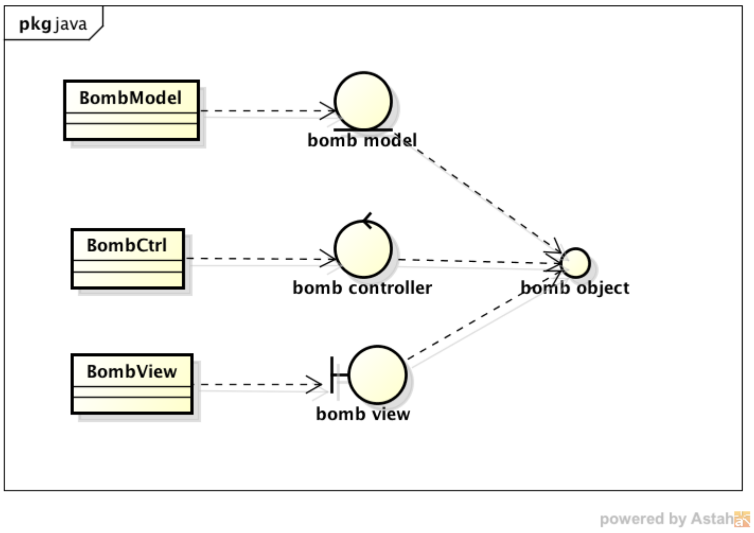


Fig 4-3

Fig 4-3 shows how MVC pattern is applied in object layer. Taking one bomb object in game as example, it is handling by three entities – a bomb model, a bomb view as well as a bomb controller. The bomb model is an instance of BombModel Class, with methods modifying attributes of bombs. The bomb view is an instance of BombView Class, with methods presenting images. The bomb controller is an instance of BombCtrl Class, with methods to handle events. Therefore, for each game object, three instances are created.

The advantages of MVC structure are obvious. First, this special MVC structure indeed supplies a clear separation of codes in terms of attribute modification, image presentation and event handling. Second, it effectively avoids undesirable mixing of attribute elements and UI elements. Third, it makes switching between different views become possible.

The drawbacks of this structure are also cannot be ignored. First, because handling one game object means to create three instances, compared to the former one-class structure, two more instances are created. If the game involved with large amount of objects, handling too much objects simultaneously would definitely overwhelm the browser, causing severe lagging or shut-down. Second, many methods in classes like BombView, RoleView, BaffleView are similar, which are regarded to be code redundancy. A preferable approach is to aggregate these view classes into one class.

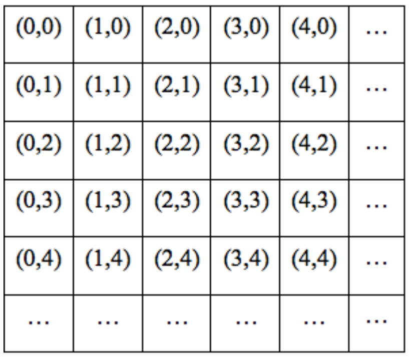
5 Functional Detail Design

This section is aiming at providing the functional design detail from a theoretical perspective.

5.1 Grid and coordinate axis system

Grid and coordinate axis system is the foundation of establishing our grid-based Popoman.

As is mentioned before the whole battle arena can be regarded as constituted by grid. Different game objects such as roles, baffles, bombs and props can be put inside the grids.

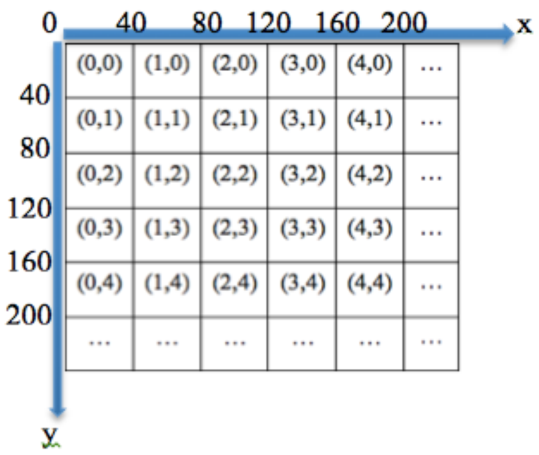


(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	…
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	…
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	…
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	…
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	…
…	…	…	…	…	…

Fig 5-1

In order to efficiently represent grids, a regular coordinate axis system is established. In this i-j coordinate system, every grid is given a unique coordinate (i, j). As shown in Fig 5-1, The coordinate for the grid in the upper left corner is (0, 0). When the battle arena is created, the game object would put inside the grids. The relationship of game object and grid is many-to-one relationship. For example, suppose there can be two roles standing in the (1, 0) grid simultaneously, which means both of them belong to grid (1, 0) at that moment. When the role moves to another grid, it is said that the role belongs to that new grid.

Besides the i-j coordinate system, Fig 5-2 shows another x-y coordinate system. Each grid is a 40 x 40 pixel square on the screen. Each point on the screen is given a unique (x, y) coordinate. The x-y coordinate axis system can help track the exactly position of the role in our game.



(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	…
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	…
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	…
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	…
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	…
…	…	…	…	…	…

Fig 5-2

5.2 Role Movement

5.2.1 Central point position

Central point position is useful to represent the position of role.

As we know, the central point of a square is its intersection of two diagonals. If we replace the role image by a square, it is reasonable to represent his position using the central point position. Here is a role in Fig 5-3. His image is replaced by a red square with the equal size of the grid. His position would be represented by its central point position.

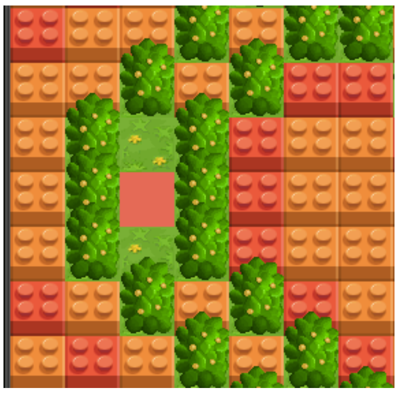


Fig 5-3

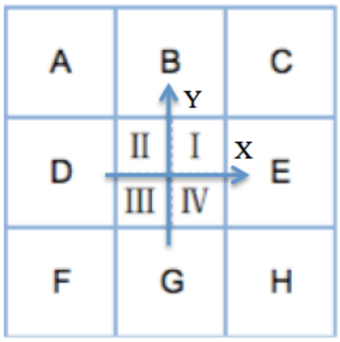
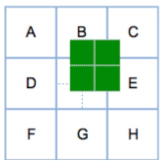
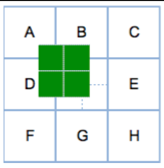
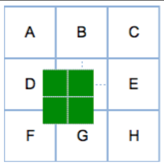
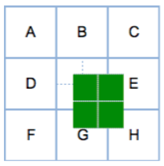
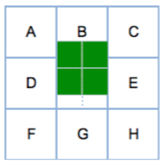
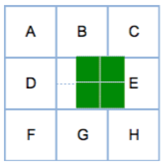
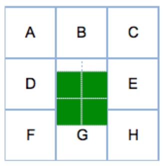
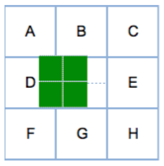
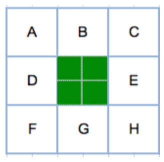


Fig 5-4

Now if we take the central point of a grid as origin and establish a regular coordinate axis system inside a grid, we get four quadrants Ⅰ, Ⅱ, Ⅲ and Ⅳ (Fig 5-4). If we try to place the role square image inside the grid, there might be night different situations. These night positions would be useful to explain the collision detection and role movement logic.

Situation	Description	Position Name	Figure
1	In the quadrant Ⅰ	pC	
2	In the quadrant Ⅱ	pA	

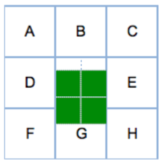
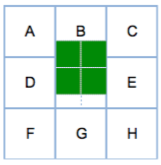
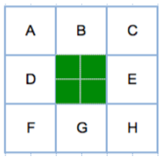
3	In the quadrant Ⅲ	pF	
4	In the quadrant Ⅳ	pH	
5	On the positive Y axis	pB	
6	On the positive X axis	pE	
7	On the negative Y axis	pG	
8	On the negative X axis	pD	
9	On the origin	pO	

5.2.2 Collision detection and handling

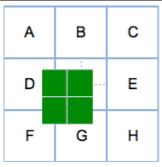
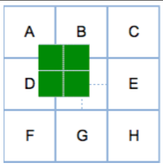
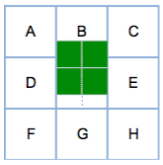
In this section, the collision detection between role and baffle is discussed. Collision detection assists the role to move properly, avoiding inaccurate overlap of images.

In Popoman, two kinds of collisions are defined.

Front collision, which is a collision that causing a stop of the original movement. In this situation, the role is stopped by obstacle and staying at critical position after the collision happens. Suppose in Fig 5-5, only grid B is occupied by a baffle. Now the attempt at moving from pG(Fig 5-5) to pB(Fig 5-6) would fail due to a front collision. The result would be pO(Fig 5-7), where the role is stopped by baffle.

 Fig 5-5	 Fig 5-6	 Fig 5-7

Side collision, which is a collision that causing a deviation of the original movement. In this situation, the role is still capable of moving ahead but a side translation has happened. Suppose in Fig 5-8, only grid A is occupied by baffle. The attempt at moving from pF(Fig 5-8) to pA(Fig 5-9) would fail due to a side collision. The result would be pB(Fig 5-10), where the role reach the expected vertical position, but has a tiny translation horizontally.

 Fig 5-8	 Fig 5-9	 Fig 5-10

Simulation is adopted in collision detection. First, a raw position is calculated merely based on direction and speed. Second, the terrain factor is considered, examining whether the adjacent grid is occupied by baffle or bomb. Third, judgment is made and the position is calibrated.

5.2.3 Key state tracking

It has been discussed in 3.1.3 Player Interaction Evaluation that a key state tracking mechanism is essential for our game, aiming at achieving a preferable control of role movement.

Left	Right	Up	Down
Yes	No	No	No

Fig 5-11

Left	Right	Up	Down
YES	NO	YES	NO

Fig 5-12

In order to keep tracking of the key state, it is essential to build a container to store the state of key. In Fig 5-11, four state-related elements are stored in the container. They are the state of the arrow keys. YES means the corresponding arrow key is pressed currently. Here the Left key is pressed, while the Right, Up and Down key are not. Thus the role moves to the left. State-related elements are modified if player press or release certain arrow key. For example, if player now press the Up, the container will change to Fig 5-12, indicates the Left and the Up key are both pressed currently by player. Because the Up is pressed later, the role moves to the up. Now if player release the Up, container will change back to Fig 5-11, because state of the Left key is YES, the role moves to the left again. Therefore, the key state tracking is done with the help of container.

5.3 Bomb Explosion

5.3.1 Building of Explosion Range

Bomb explosion range is a set of grids that are covered by bomb power, exposed to the water spray. In Fig 5-13, there are a Level-2 bomb, a brick and a house. All the red squares are grids that belong to the explosion range.

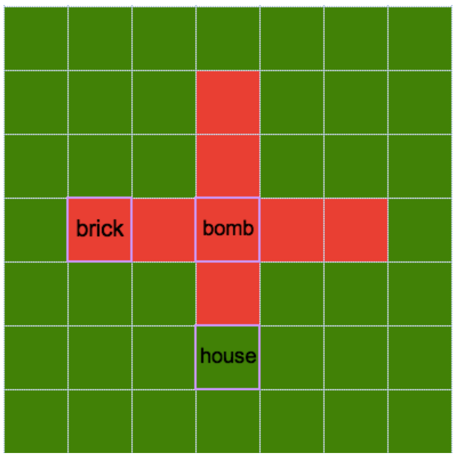


Fig 5-13

As the bomb explodes, it release spray both vertically and horizontally. Bomb explosion range should be created before bomb explosion. The steps to build the explosion range are as follow. 1. Start with the bomb, seeking for new adjacent grid both vertically and horizontally. If the grid found is empty, add it to the explosion range. If the grid found is occupied by house baffle, stop the seeking in current direction. If the grid found is occupied by brick baffle, stop seeking and add this grid to the explosion range. After examining all the grids in left, right, up and down directions, the final explosion range is built.

5.3.2 Successive Explosion

Bomb explosion in Popoman is in a successive approach. Spray released by any exploding bomb can detonate other bomb. In Fig 5-14, there are a Level-1 bomb1 and a Level-2 bomb2. Red regions are their respective explosion range.

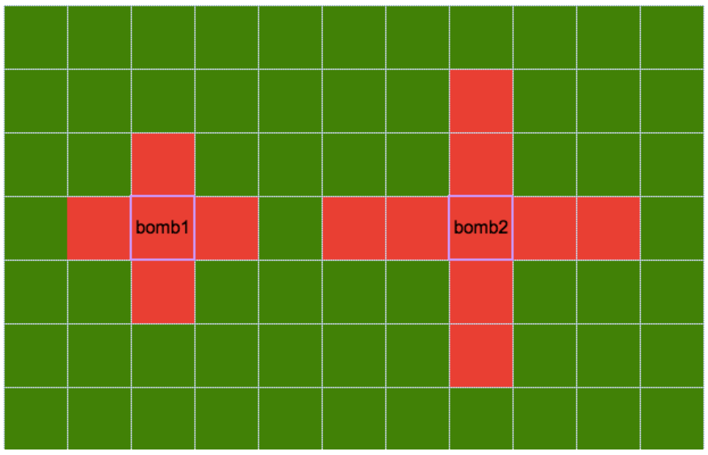


Fig 5-14

In Fig 5-15 and Fig 5-16, they are placed closed to each other, which causes an overlap of explosion range. Suppose bomb2 is placed later than bomb1 on the arena, in this case, bomb2 would be detonated by bomb1 and they together create a new explosion range.

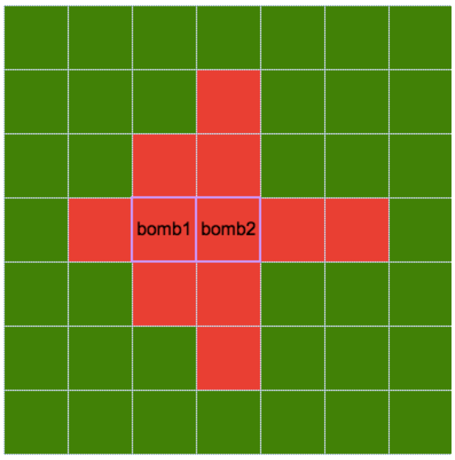


Fig 5-15

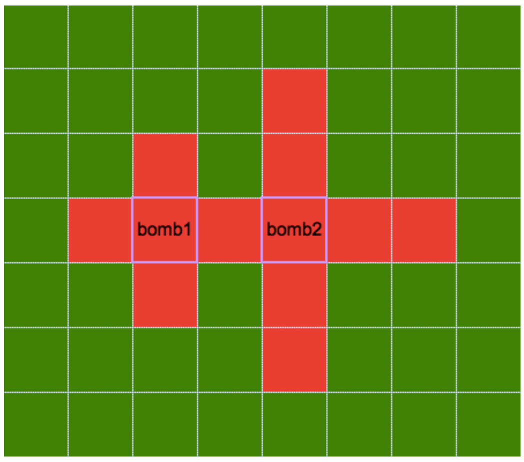


Fig 5-16

To achieve this successive explosion effect, each bomb should contain a timer to handle its timing of explosion. For example, the timer is set to be 2 seconds in default, which means each bomb released on the arena would explode in 2 seconds. Then if a bomb is detonated by spray, the timer is immediately set to be zero, which means it is the right time for the bomb to explode. Therefore it ensures all the bombs with timer equal to zero explode simultaneously.

5.4 Game-AI

5.4.1 Safe Grid & Risky Grid

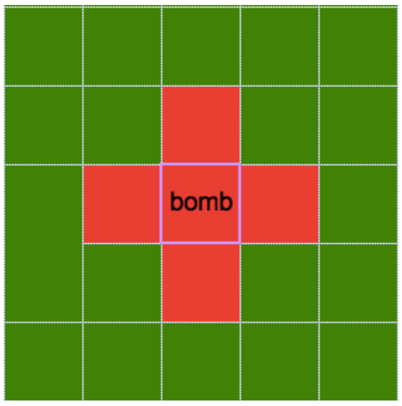


Fig 5-17

In Fig 5-17, a bomb is released on the arena. The red squares are the explosion range. Here we define two types of grids.

1. Risky grid. The grid currently belongs to any bomb explosion range.
2. Safe grid. The grid currently not belongs to any bomb explosion range.

We use red squares to stand for the risky grids, while green squares stand for the safe grids. To name it 'Risky' is to emphasize the risk of walking through such kind of grid. Because the exact explosion time is unknown, role that attempts to walk through a risky gird might be exposed to a potential risk to be killed by unexpected explosion. Risky grid and safe grid reflect the safety of grid. They are the criteria of game-AI movement.

5.4.2 Safe Path & Risky Path & Death Path

A path is comprised of a set of successive girds in one direction. Here we define three types of paths.

1. Safe path: A path that only contains safe grid.
2. Risky path: A path that contains both safe grid and risky grid.
3. Death path: A path that either contains on grid or only contains risky grid.

In Fig 5-18, the grids inside the yellow lines indicate the path in each direction – the left path, the right path, the up path and the down path.

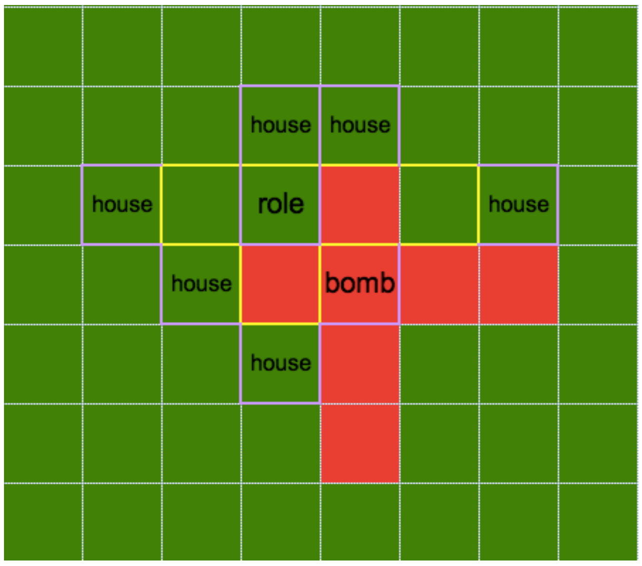


Fig 5-18

From the definition, we can draw a conclusion:

1. The left path is a safe path because it only contains safe grid.
2. The right path is a risky path because it contains a safe grid as well as a risky grid.
3. The up path is a death path because it contains no grid.
4. The down path is a death path because it only contains risky grid.

5.4.3 Path-finding algorithm

The path-finding algorithm is based on three types of paths defined in former section. Several path-finding algorithms are proposed varied from their complexity.

Algorithm01: It merely based on priority processing of different types of paths.

Priority	Situation	Result
High	Safe path = 1	This direction
	Safe path > 1	Random direction
Low	Risky path = 1	This direction
	Risky path > 1	Random direction
Zero	Death path	Ignored

The safe path is given a higher priority, while the risky path is given a lower priority. As for death path, it would be omitted by the game-AI.

For example, in Fig 5-18 situation, the game-AI will directly choose the left path because it is a safe path. As for in special case, if two or more safe paths exist simultaneously, the new direction will be randomly generated out of them, so do risky paths.

But the situations of risky paths are more complicated than safe paths. As shown in Fig 5-19, there are two risky paths (right and down) and two death paths (left and up). According to **Algorithm01**, game-AI will randomly make a choice between right and down. However, as the Fig 5-19 shows, the distance to the nearest safe grid in right path is two, while in down path is three. It is definitely more reasonable for game-AI to choose the right path rather than make a random choice. Therefore we need a more scientific algorithm to handle it.

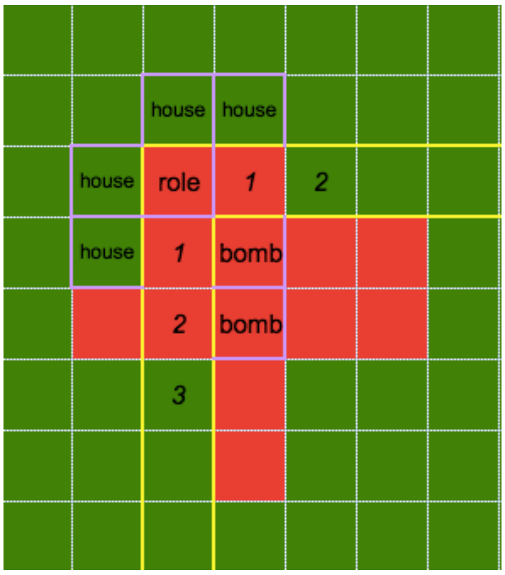


Fig 5-19

Algorithm02: It is based on Algorithm01, the distance factor is considered when judging risky paths.

Priority	Situation	Result
High	Safe path = 1	This direction
	Safe path > 1	Random direction
Low	Risky path = 1	This direction
	Risky path > 1	The direction with the nearest safe grid
Zero	Death path	Ignored

6 Implementation Description

6.1 Screens

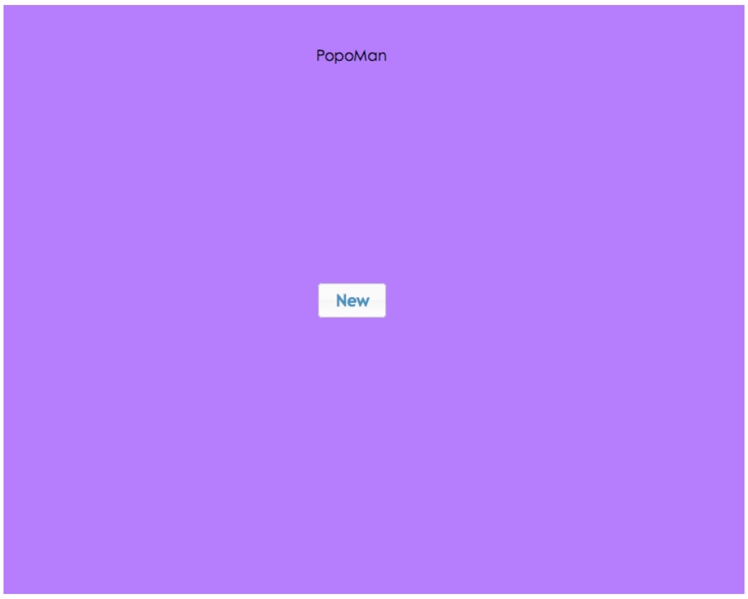


Fig 6-1

In Popoman, there are totally five screens. Despite of the game screen, other four screens are basically constituted by a screen div element, a text element and some button elements. As we know, the text and button are both embedded inside the screen. It is useful to apply the DOM approach to handle their relationship. Taking splash screen (Fig 6-1) for example, the following is the primary implementation in Game Class.

```
Game.rootWindow = function(){
    UI.div("rootWindow");
    UI.text("rootText","rootWindow","PopoMan");
    UI.button("newButton","rootWindow","New");
    document.getElementById("newButton").addEventListener("click",function(){
                    Window.remove("rootWindow");
                    Window.show("modeWindow");
                },false);
}
```

When the game starts, the above method is automatically run by the game. First, the screen is built by using the UI method to create a div element on the screen. Then a text element and a button element are embedded into the screen div element by using JavaScript DOM method – appendChild . Next, an event listener is added to the button, which listened to button click event and switch to the mode selection screen.

```
1  UI.text = function(text_id,father_id,innerText){
2      var text = document.createElement("div");
3      document.getElementById(father_id).appendChild(text);
4      text.setAttribute("id",text_id);
5      text.innerHTML = innerText;
6  };
```

Line-2 indicates the text is a div element that contains the text. Line-3 uses the appendChild method to add this text element into the DOM tree. In this case, the text div element becomes the child element of the screen div element.

The advantage of using DOM approach is if the father element in DOM tree is removed, all of its associated child elements are also removed.

6.2 Arena Building

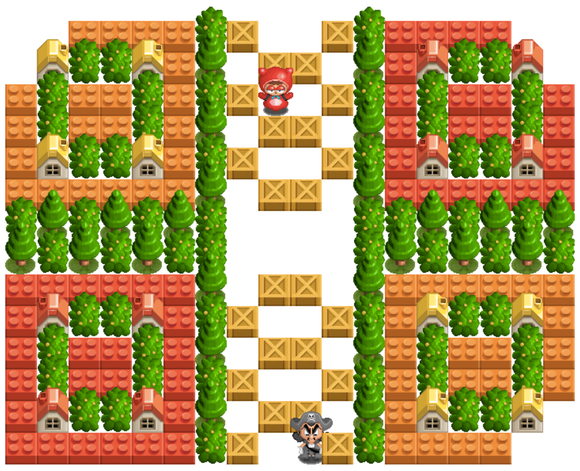
Building a grid map in JavaScript has several approaches. The format to represent the map can be array, JSON or XML. The final decision is to use array because of its easy handling feature. To be specific, the grid map is simulated by two-dimensional array.

In Popoman, the arena is comprised by a background stage and all the game objects including roles and baffles. Two arrays are used. The first array contains the tiles to build the background stage of the arena. The second array contains all the game objects including roles and baffles.

The multiDistrict[] array will map all the game objects on the arena. (Fig 6-2) The Integer number in the multiDistrict[] array is associated with the element in type[] array.

```
Map.type = [
    'none','bric1', 'bric2', 'house1', 'house2', 'house3', 'box',
    'tree1','hideWall','tree2','leaf','log','rock1','rock2'
];
Map.multiDistrict = [
    [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
    [-1, 0, 0, 1, 1, 1, 1, 8, 6, 0, 0, 6, 8, 2, 2, 2, 2, 0, 0,-1],
                          . . . . . .
                          . . . . . .

    [-1, 2, 2, 2, 2, 2, 2, 8, 6, 0, 0, 6, 8, 1, 1, 1, 1, 0, 0,-1],
    [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
];
```



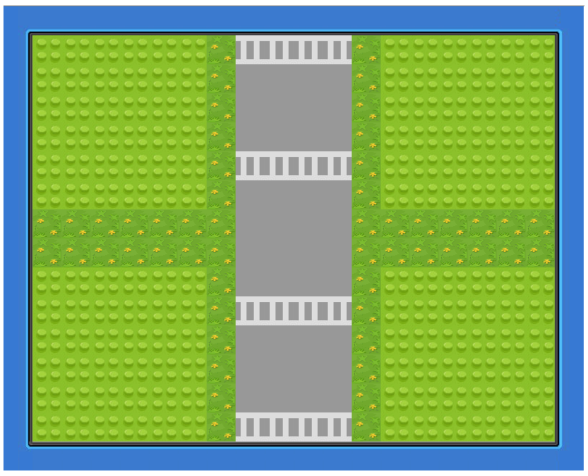


Fig 6-2 **Fig 6-3**

DistrictBaseMap[] array working with baseType[] array to build the background in Fig 6-3.

```
Map.baseType = [
            'border_lt','border_t','border_rt','border_r',
            'border_rb','border_b','border_lb','border_l',
            'yellowGround','greenGround','brownGround',
            'greyGround','crossGround','bricGround'
];

Map.DistrictBaseMap = [
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2],
    [7,13,13,13,13,13,13, 9,12,12,12,12, 9,13,13,13,13,13,13, 3],
    [7,13,13,13,13,13,13, 9,11,11,11,11, 9,13,13,13,13,13,13, 3],
                               .......
    [7,13,13,13,13,13,13, 9,11,11,11,11, 9,13,13,13,13,13,13, 3],
    [7,13,13,13,13,13,13, 9,12,12,12,12, 9,13,13,13,13,13,13, 3],
    [6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4],
];
```

Together they build the arena (Fig 6-4) with background stage as well as all the roles and baffles on it.

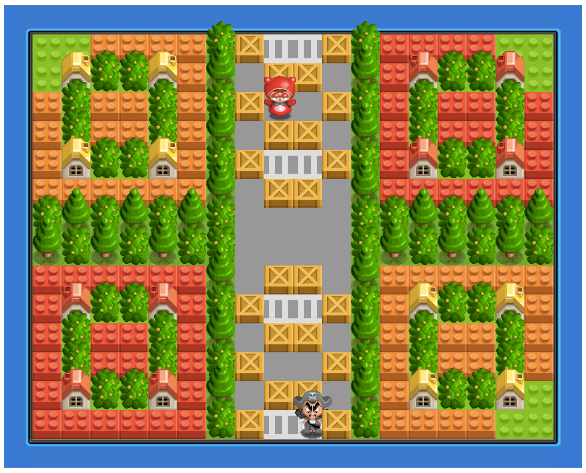


Fig 6-4

6.3 MVC pattern in Object layer

In Popoman, MVC is adopted in an object layer. In Game Class, as the arena is created, we created three objects – a model, a view and a controller, then the model and view are passed as parameters to the controller constructor.

```
                  var mainRoleModel = new RoleModel(i,j,'mainRole');
                  var mainRoleView = new RoleView();
                  var mainRoleCtrl = new RoleCtrl(mainRoleModel,mainRoleView);
```

In RoleCtrl Class, the model and view object is stored as the internal elements of controller.

```
function RoleCtrl(roleModel, roleView){
    this._roleModel = roleModel;
    this._roleView = roleView;
}
```

Therefore, the controller can access its corresponding model and view, and call the appropriate methods. In *move* method, controller calls the model to handle the data calculation, where the results are sent back, wrapped with an array. Next, controller updates the view with the result data.

```
RoleCtrl.prototype.move = function(){
    var arr = this._roleModel.move();
    var newX = arr[0];
    var newY = arr[1];
    var j = arr[2];
    var state = arr[3];
    var direction = arr[4];
    var type = arr[5];
    this._roleView.move(newX, newY, j, state, direction, type);
}
```

RoleModel Class is entirely focusing on data calculation. Such as the following *placeBomb* method, it judges whether the role is able to release a bomb successfully in current position, and it returns the flag to the controller.

```
RoleModel.prototype.placeBomb = function(){
    var i = this.getI();
    var j = this.getJ();
    var type = this.getType();
    var gird = Game.map[j][i];
    var bombAvailable = this.getBombAvailable();
    var flag = false;
    if(gird.bomb==null && bombAvailable>0){
        this.getBombList().pop().play();
        flag = true;
    }
    return flag;
}
```

RoleView Class is entirely responsible of the data presentation. Such as the following *move* method, which manipulates the related CSS attributes to control the position of the image, and it also makes the decision on showing the proper image according to current state.

```
RoleView.prototype.move = function(x, y, j, state, direction, type){
    this.dom.style.left = x - Game.WIDTH/2;
    this.dom.style.top = y + Game.HEIGHT/2 - Picture[type+direction+state].height;
    this.dom.style.zIndex = j;
    if(this.img.src != Picture[type+direction+state].src){
        this.img.src = Picture[type+direction+state].src;
    }
};
```

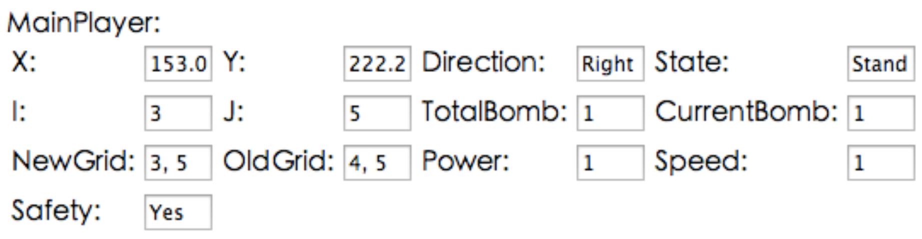
7 Test and Evaluation

7.1 Debugging Process – Using a Test-panel

This section is aiming at showing the real process how the debugging process was done during the development.

Due to a lacking of real experience in programming, the codes are not developed using TDD (Test-Driven-Development). Therefore, the approach to conduct unit testing automatically by a testing tool cannot be realized. The debugging is primarily conducted manually based on the console. As a matter of fact, the test cases for manual testing were not well written before the implementation phase. Thus the debugging process was not so traceable.

In order to make the manual debugging more efficient, I built a test-panel beside the game area on the browser. It reflects the underlying data of the role. It contains the position, direction, state, bomb amount, bomb power level, speed and safety. Test-panel visualizes the intrinsic states that might not be tested from watching the external behavior. Here are a screenshot of the test-panel for main player. There is another one for sub player. (Fig 7-1)



MainPlayer:

X:	153.0	Y:	222.2	Direction:	Right	State:	Stand
I:	3	J:	5	TotalBomb:	1	CurrentBomb:	1
NewGrid:	3, 5	OldGrid:	4, 5	Power:	1	Speed:	1
Safety:	Yes						

Fig 7-1

Terms	Description
X	X position of role in x-y coordinate system
Y	Y position of role in x-y coordinate system
Direction	Current direction the role face to
State	Indicate whether the role is moving or still.
I	I position of role in i-j coordinate system
J	J position of role in i-j coordinate system
TotalBomb	Number of total bombs carried by role

CurrentBomb	Number of bomb can be released currently by role
NewGrid	The current grid containing the role
OldGrid	The last grid containing the role
Power	Power level of role
Speed	Speed level of role
Safety	State of grid that the role standing on

7.2 Acceptance Testing

The acceptance testing is based on the test cases. The testers conduct a series of actions on the test cases to examine whether the game is developed properly.

On the splash screen, testers can click the Test button enter the test screen. Here is a test map. Testers can conduct test with the guidance of test cases.

Test Case Template：

Test Case	Test case id. (A.TC001, B.TC002, C.TC003, etc….)
Description	A short description of the objective of this test case.
Steps	The steps for tester to conduct sequentially.
Expected Result	Expected results of the test.
Status	FAIL / PASS
Related Requirement	The related requirement (RS001,RS002, etc…)

Test Case Board:

Test Suits		**Test Cases**	
A	Role Movement	A.TC001	Grid-free Moving Style
		A.TC002	Collision Detection and Handling
		A.TC003	Key State Tracking Mechanism

B	Bomb Event	B.TC001	Bomb Release Event
		B.TC002	Bomb Explosion Event (single)
		B.TC003	Bomb Explosion Event (multiple)
		B.TC004	Bomb Destroy Event
C	Prop Event	C.TC001	Prop Generation Event
		C.TC002	Prop Collection Event
D	Sound	D.TC001	System Sound
E	Game Logic	E.TC001	Victory Condition
F	Menu UI	F.TC001	Button Functionality
G	Game-AI	G.TC001	(unfinished)
H	Compatibility	H.TC001	Cross-browser Compatibility
I	Privacy	I.TC001	Source Code Privacy

Test Suit A: Role movement

Test Case	A.TC001
Description	Grid-free moving style
Steps	1. Move the role randomly in each direction.
Expected Result	1. Role can move freely, stop at any position, without restriction of grid.
Status	All PASS
Related Requirement	1) RM.RS001

Test Case	A.TC002
Description	Collision detection and handling
Steps	1. Move the role to across the border of arena. 2. Move the role to walk through brick, house, box and tree.

	3. Move the role to walk through jungle and role. 4. Release a bomb, and move the role to walk through bomb. 5. Move the role to make front collision. 6. Move the role to make side collision.
Expected Result	1. Cannot move outside the arena. 2. Cannot walk through brick, house, box and tree. 3. Can walk through jungle and role. 4. Cannot walk through bomb. 5. Be stopped, cannot move on. 6. Can move on but with a side translation.
Status	All PASS
Related Requirement	1) RM.RS002 2) RM.RS003 3) RM.RS004 4) RM.RS005 5) RM.RS006

Test Case	A.TC003
Description	Key state tracking mechanism
Steps	1. Press Left -> press Right-> release Right. 2. Press Up -> press Down -> release Down. 3. Press Left -> press Up -> release Up. 4. Press Right -> press Down -> release Down.
Expected Result	1. Role moves to: Left -> Right -> Left 2. Role moves to: Up -> Down -> Up 3. Role moves to: Left -> Up -> Left 4. Role moves to: Right -> Down -> Right
Status	All Pass
Related Requirement	1) RM.RS007

Test Suit B: Bomb Event

Test Case	B.TC001
Description	Bomb release event
Steps	1. When having remaining bombs, move to an empty grid. Press Release key once to release a bomb. 2. When run out of bomb, move to an empty grid. Press Release

	key once to release a bomb. 3. When having two more remaining bombs, press Release key for twice, without moving.
Expected Result	1. A bomb is released, placed on the current grid. Remaining bombs reduced by 1. 2. Cannot release bomb, because all the bombs have been released already. 3. Only one bomb is released, because each grid only can hold one bomb.
Status	ALL PASS
Related Requirement	1) B.RS001 2) B.RS002 3) B.RS003

Test Case	B.TC002
Description	Bomb explosion event (single)
Steps	1. Release a bomb. 2. Release a bomb closed to border. 3. Release a bomb closed to tree, house, box and brick. 4. Release a bomb closed to jungle andƒ role. 5. Release a bomb and watch the bomb level.
Expected Result	1. Bomb can explode after two seconds. 2. The spray is blocked by border. 3. The spray cannot propagate through tree, house, box, and brick. 4. The spray can propagate through jungle and role. 5. Bomb level should equal to [Power]
Status	ALL PASS
Related Requirement	1) B.RS004 2) B.RS005 3) B.RS006 4) B.RS007 5) B.RS008

Test Case	B.TC003
Description	Bomb explosion event (multiple)
Steps	1. Release one bomb, then release another bomb within its explosion range.

	2. Release one bomb, then release another bomb outside its explosion range.
Expected Result	1. The two bombs explode simultaneously. 2. The two bombs explode respectively.
Status	ALL PASS
Related Requirement	1) B.RS009

Test Case	B.TC004
Description	Bomb destroy event
Steps	1. Release a bomb closed to role. 2. Release a bomb closed to tree and house. 3. Release a bomb closed to box, brick and jungle.
Expected Result	1. Role is destroyed. 2. Tree and house cannot be destroyed. 3. Box, brick and jungle can be destroyed.
Status	ALL PASS
Related Requirement	1) B.RS010 2) B.RS011 3) B.RS012

Test Suit C: Prop Event

Test Case	C.TC001
Description	Prop generation event
Steps	1. Release a bomb closed to box, brick and jungle.
Expected Result	1. Box, brick and jungle can be destroyed, and randomly generate a kind of prop.
Status	ALL PASS
Related Requirement	1) P.RS001

Test Case	C.TC002
Description	Prop collection event
Steps	1. When [TotalBomb]!=Max, move the role to collect a bomb-up. Then release bombs. 2. When [TotalBomb]==Max, move the role to collect a bomb-up. Then release bombs. 3. When [Power]!=Max, move the role to collect a power-up. Then release a bomb. 4. When [Power]==Max, move the role to collect a power-up. Then release a bomb. 5. When [Speed]!=Max, move the role to collect a speed-up. 6. When [Speed]==Max, move the role to collect a speed-up.
Expected Result	1. Bomb-up is collected, [TotalBomb]++. 2. Bomb-up is collected, [TotalBomb]==Max. 3. Power-up is collected,[Power]++. 4. Power-up is collected,[Power]==Max. 5. Speed-up is collected,[Speed]++. 6. Speed-up is collected,[Speed]==Max.
Status	ALL PASS
Related Requirement	1) P.RS002 2) P.RS003 3) P.RS004

Test Suit D: Sound

Test Case	D.TC001
Description	System Sound
Steps	1. Listen when stay at the main menu. (Start from splash screen, navigating to mode screen and map screen.) 2. Listen when begin a new round. 3. Listen when bomb released. 4. Listen when bomb explode. 5. Listen when prop collected. 6. Listen when box pushed. 7. Listen when role killed. 8. Listen during the whole round. 9. Listen when the result is WIN. 10. Listen when the result is LOSE. 11. Listen when the result is DRAW.

Expected Result	1. A BGM should be played and looped. And the BGM should not begin again when screen switching happens. 2. A sound should be played when a new round begin. 3. A sound should be played when bomb released. 4. A sound should be played when bomb explode. 5. A sound should be played when prop collected. 6. A sound should be played when box pushed. 7. A sound should be played when role killed. 8. A BGM should be played and looped during the whole round. 9. A sound should be played when the result is WIN. 10. A sound should be played when the result is LOSE. 11. A sound should be played when the result is DRAW.
Status	ALL PASS
Related Requirement	1) S.RS001 2) S.RS002 3) S.RS003 4) S.RS004 5) S.RS005 6) S.RS006 7) S.RS007 8) S.RS008 9) S.RS001 10) S.RS010 11) S.RS011

Test Suit E: Game Logic

Test Case	E.TC001
Description	Victory Condition
Steps	1. Play the solo mode, and kill the game-AI opponent within the playtime. 2. Play the solo mode, and suicide within the playtime. 3. Play the multi mode, and let main-Role kill the sub-Role within the playtime. 4. Play the multi mode, and let sub-Role kill the main-Role within the playtime. 5. Play the multi mode, and keep both main-role and sub-role alive until the playtime runs out.
Expected Result	1. Result is 'You Win' 2. Result is 'You Lose' 3. Result is 'Main Role Win'

	4. Result is ‘Sub Role Win’ 5. Result is ‘Draw’
Status	ALL PASS
Related Requirement	1) GL.RS001 2) GL.RS002 3) GL.RS003 4) GL.RS004 5) GL.RS005

Test Suit F: Menu UI

Test Case	F.TC001
Description	Button Functionality
Steps	1. In the splash screen, click NEW. 2. In the mode screen, click SOLO. 3. In the mode screen, click Multi. 4. In the map screen, select a map. 5. In the map screen, click GO.
Expected Result	1. Should enter mode screen. 2. Should enter map screen. 3. Should enter map screen. 4. The selection menu is working. 5. New round begins, with the correct mode and map.
Status	ALL PASS
Related Requirement	1) UI.RS001 2) UI.RS002 3) UI.RS003 4) UI.RS004 5) UI.RS005

Test Suit G: Game-AI

This part is missing due to difficult to derive an effective testing strategy to exactly test the game-AI behavior.

Test Suit H: Compatibility

Test Case	H.TC001
Description	Cross-browser compatibility

Steps	1. Run the game in Chrome, perform the test in Test Suite A – G. 2. Run the game in Firefox, perform the test in Test Suite A – G. 3. Run the game in Safari, perform the test in Test Suite A – G.
Expected Result	1. The game run smoothly, without functionality defect. 2. The game run smoothly, without functionality defect. 3. The game run smoothly, without functionality defect.
Status	PASS (1, 2) FAIL (3) Details: The game suffers substantial lagging in Safari, possibly due to the poor support for GIF image rendering in Safari.
Related Requirement	1) NF.RS001

Test Case	I.TC001
Description	Source Code Privacy
Steps	1. Run the game and click the right button of mouse.
Expected Result	1. The context menu is disabled.
Status	PASS
Related Requirement	2) NF.RS002

8 Conclusion And Outlook

The thesis focuses on showing the capability of JavaScript language by building a variation of an existing game in a scientific software development approach.

This thesis demonstrates some parts of Software Development Life Cycle (SDLC) such as Requirement Analysis, Design, Implementation, Testing and Evaluation. The first phase is Requirement Analysis, in which the game concept of the original game Bomberman is analyzed. The result is the requirement specification of the target game – Popoman. The second phase is Design, in which the system architecture as well as the desired features the operation details is designed. The third phase is Implementation, in which the real code is real code is written. The Test and Evaluation is not an isolated phase, but should be conducted simultaneously with the Implementation phase. The test should be conducted under the guidance of the test strategy, usually a concrete detailed test case document.

In this project, less effort was spent on the Requirement Analysis, Design and Test at first, which led to poor communication and corpulent architecture. Although the Implementation part was conducted successfully, for the derivable game was interesting to play and most of the requirements are met. From this real practice, the author gained a deep insight into the JavaScript game programming, and realized the significant importance of the Requirement Analysis, Design and Test.

In the future, the author will conduct the SDLC in a more scientific way. As for Popoman, it can be redesigned, adopting a better architecture to avoid code redundancy, and develop a more sophisticated AI algorithm.

Acknowledgments

Here the author wants to thanks the mentor – Prof. Nane Kratzke. Thanks for instructing the thesis including providing feedback and help.

Bibliography

[1] Wikipedia.Bomberman [Online] http://en.wikipedia.org/wiki/Bomberman

[2] Wikipedia.JavaScript [Online] http://en.wikipedia.org/wiki/JavaScript

[3] Douglas Crockford. (2001) JavaScript: The World's Most Misunderstood Programming Language [Online] http://www.crockford.com/javascript/javascript.html

[4] Jennifer Kyrnin.Why Use HTML5 Canvas.[Online]

http://webdesign.about.com/od/html5tags/a/why-use-html5-canvas.htm

[5]Wikipedia. Document Object Model.[Online]

http://en.wikipedia.org/wiki/Document_Object_Model

[6]Wikipedia. Canvas element.[Online] http://en.wikipedia.org/wiki/Canvas_element