# Code Vs. The Design Document

Frinkiac-7

November 22, 2004

**Abstract**

This document explains the discrepancies between the design document and the final code that will go with the game product for bomberman. The code has been excessively modified from the version in the design document althought the basic structure still holds. Many new functionality was added and classes to clean up the original code. As the coding went along it was seen beneficial to do this to keep the code clear and concise. This document holds the explanations and reasons of the decisions made concerning the code.

## Contents

## 1 Class Widget

In widget there were many variables added: frames, frameIndex, frameRate, timeDelay, lastUpdate, world, undone, and an id. These were added to improve animation and reduce the amount of redrawing to only when the widget has been updated by a movement. Consequently functions needed to be added to do this: attachToWorld(), startAnimation(), and getShadow(). The id is used to keep track of all the objects when needed for sorting and to take care of nondeterminism.

## 2 Class WorldlessWidget

This was a class that was added to move and animate a widget that is not attached to the world.

## 3 Class Bomb

As the code got more complicated and the game needed to do more with the bomb then just allow a bomb to be dropped, there had to be many additional variables added to bomb. These include: lastMoved, napalm, radius, beingPunched, placed, timeDelay, last_update, frameIndex, modNo, baseFrame, and image. Mainly as the power-ups were implemented we needed to be able to track them on the bomb as the bomberman placed them. There is also an increased number of functions required to work with the bomb. The first additional is snapToGrid(). Now that the bomberman is capable of walking inbetween the gridsquares rather than from square to square, when he places the bomb, it takes his position inbetween the squares. Therefore in order to fix this problem we had to add a function snapToGrid that places the bomb he placed in the middle of the grid. Functions to place the bomb had to be created to keep a list that would be needed to check on players walking over a bomb, which shouldn't happen. The update() function was added for animation purposes to actually show the ticking of the bomb and its count down to blowing up. This function also keeps track of how the bomb is being placed and where and keeps if from being placed in places where it will cause a collision. The tick function was no longer needed because the decrement of the TTL was enough to keep track of the time to blow up and the actual explosion does not occur within this class any longer.

## 4 Class Bomber

This was once known as Bomberman, but that is the name to run the game. Bomber is the actual widget of the Bomberman. This class changed a great deal once all the power-ups were complete. Variables included recognizing whether that particular bomberman player had any viruses or power-ups. Variables such as name, color, and player had to be added to keep track of the bomberman now that the players could pick a color and add their names. This way which bomberman dropped a bomb could be kept track of, which died, which won, etc. The default values were set for speed, radius, BOMBTTL, and so on before the powerups were calculated and as the power-ups are calculated these values would change. There were also variables added for animation to face the player in the direction that he is moving. CcreateFrames(), animatePlayer() and faceBomber() are functions that were added for animation. In addition to all the variables added for the power-ups, with them had to come functions. Each power-up has at least one function to keep track of what it does to the bomberman, for example the setPunch() which deactivates the detonateEnable if they had it and allows the punch to happen. The LifeUp() is another example that increases the number of lives of the bomberman.The virus functions are also all included in here.

## 5 Class Explosion

Not too many changes occured here. A couple of variables were added because the addition of napalm requires not only different bombs, but it is a different kind of explosion and does not act like normal ones. Normal ones destroy powerups and viruses, the napalm doesn't. Variables also had to be added for animation and of course adding an ID to keep track of what is in the gameworld. We also added sound to the game so of course functions to implement sound of the explosion had to be added. The update function() which is in almost every class to update the environment after something occurs had to be added to take care of changes after an explosion and because powerups are created when the explosion destroys muttable objects then a createPowerUp() had to be added to do so.

## 6 Class PowerUp

All power-ups whether viruses or powerups were placed into one class. They are not different in functionality. They do either good or bad but that was not enough to differentiate them into two classes, so they were combined into one. Sound functions were added to happen when a bomberman would pickup a power up. The activate() function was improved and now sets all the details for the powerups which now calls the functions set in the Bomber Class that set the appropriate variables. Therefore there is a missing association from our design document between class Bomber and class PowerUp. And of course update() was added.

## 7 Input

As to our clients request, joysticks were added to the game to control the bomberman. Therefore, our keyboard input Class changed to the base class InputDevice. Its subclasses were a Keyboard class class and a Joystick class, for keyboard and joystick input.

## 8 Color Picker

ColorPicker is still used, but its visual UI is no longer needed. It has been replaced by the new class, BombermanUI, because our client requested a better menu in our game. This menu was built using the wxWidget library, which was not originally included in our Deployment diagram, but which is now a required library.

## 9 class Player

The player class had to add additional variables to keep control of the name of the player and the player number in the list. The score is also kept here and the color chosen by the player at the beginning of the game. Its subclasses include local player, network player, and a scripted player. These were added to allow for network play.

## 10 class BombermanFrame

The client requested a menu to start the game. Therefore wxglade was used and the python code generated and then added upon and integrated with the rest of the code.

## 11 class GameSetup

This class was started after networking was introduced. There are two subclasses: NetworkSetup and LocalSetup. From NetworkSetup there is further inheritance to either a ServerSetup or ClientSetup. These classes initialize all game aspects depending on which type is started. They connect all the networking machines together if necessary.

## 12 class Map

A simple class used to construct the map outside of game.py. Refactoring of the code removed this step from game.py to its own class. Now game.py is more cohesive.

## 13 splashes.py

An extra module for holding our splash screens which are complicated to create. The code was removed from game.py and placed on its own as part of refactoring.

## 14 translator

Helps to initialize the joystick input.

## 15 class Game

Many variables were added to game most importantly the random seed that gets passed across the network. The function exchangeInput()centralizes and synchronizes all players input constantly especially during network play. Bomb functions, e.g. dropBomb() occur here as well as sounds, ending the game, keeping track of the winning player. This is where the main game loop is and the graphics rendering occurs.

## 16 class Gameworld

As far as variables go, added to this file was the number of mapRows and mapColumns. Once powerups were complete all appending and removing from the gameworld happens here. For example appendPowerUp() and removeExplosion(). The function worldWrap() was included after the punch powerup was added. Therefore, the bomb can be punched from right to left, top to bottom and vice versa on the screen. The bomberStrafe() was added to keep the bomberman from blowing himself up if he is randomly on the screen. This function will allow the bomber to snap to grid and avoid explosions.