TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS, PULCHOWK

BOMBERMAN

BY:
BAIBHAV BISTA (073BCT511)
LUMANTI DANGOL (073BCT521)
MOHIT KEDIA (073BCT523)

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
LALITPUR, NEPAL
2074/11/15

Acknowledgement

We would like to express our gratitude to the Department of Electronics and Computer Engineering, Pulchowk Campus for providing us platform to design and develop modern innovations to the world of computing. Similarly, we would like to thank our Object Oriented Programming lecturer Daya Sagar Baral for making us clear about the concepts on C++ and its features and his lab units for providing a platform to get acquainted to the theoretical as well as practical concepts on object oriented programming.

We can’t leave to thank our dear friends as they have provided us with their valuable comments and suggestions for the completion.

We would also like to thank all the writer of the books that we have used. Without their books, it would have been hard and tedious for us to get knowledge and information regarding the subject matter and it might have taken long time to complete this project.

Many many thanks to all of them whose direct and indirect contributions have made it possible for us to complete this project.

Abstract

This project work was given to us as a minor project for our academic session B.E. (Computer) Second Year First Part as prescribed in the syllabus designed by IOE, TU. The main aim of this project was to develop a user-friendly program using an Object-oriented Programming language, C++. For this project we plan to make a classic strategic maze-based video game “Bomberman”. The concept of the game was introduced in the early 1990s and later this game became very famous. The rule for playing this game is very easy as it doesn’t have too many rules and regulations that users have to follow. In this game the player places down bombs which explode in multiple directions that destroys obstacles or kills other players. The player is killed if they get caught up in a bomb’s explosion.

TABLE OF CONTENTS

Objectives:

The main objectives to be met in this project can be summarized as below:

- To develop a video game where a character is free to move in an adjacent empty space.
- To build obstacles in random spaces at the beginning of a level.
- To allow the player to drop a bomb whenever they want that explodes in multiple directions depending on the obstacles in the adjacent cells.
- To scatter power-ups under random obstacles that adds extra abilities to the player like speeding them and powerful explosions.
- To allow the application to remain responsive to the user input while executing the tasks in the background.
- To allow the application to take the input from two users parallelly.

Introduction:

Bomberman is a strategic, maze-based video game franchise, originally developed by Hudson Soft and currently owned by Konami. The original game was published in 1983 and new games have been published at irregular intervals ever since. Today, Bomberman has featured in over 70 different games on numerous platforms.

Bomberman games feature the grid-base action. Players are put in a stage where they have to destroy soft blocks, possibly uncovering items. This game largely revolves around two modes of play; single player campaigns where the player must defeat enemies and reach an exit to progress through levels, and multiplayer modes where players must attempt to eliminate each other and be the last one standing. Gameplay involves strategically placing down bombs, which explode in multiple directions after a certain amount of time, in order to destroy obstacles and kill enemies and other players. The player can pick up various power-ups, giving them benefits such as larger explosions or the ability to place more bombs down at a time. Power-Ups are the basic items to help Bomberman defeat opponents. Power-Ups come in different abilities, such as the ability to carry more bombs or faster movement. The player is killed if they touch an enemy or get caught up in a bomb's explosion, including their own, requiring players to be cautious of their own bomb placement. In addition to the main maze-based Bomberman games, some spin-off titles involve adventure, platformer, puzzle, and kart racing gameplay.

Multiplayer

There is a battle mode and a team mode in multiplayer. Up to four people can play. Four additional multiplayer maps can be unlocked, either by collecting all 120 Gold Cards in the story mode (permanent unlock), or by rapidly pressing the Start button very quickly on the main menu (lasts until the game is turned off)..

Battle mode

The objective in this is to blow up the other players (or throw them off of the stage) and survives. Once you die you become a ghost and can take control of

another player (that isn't already a ghost) for a brief time. Last one standing wins. Up to 4 people can play battle mode. It is very different to most Bomberman multiplayer battle modes in that the arena is quite free-roaming, rather than the usual grid based arenas, and players always have the ability to kick, throw and pump bombs. Also, certain maps have a pronounced third dimension, with stairways and multiple elevations from which bombs can be dropped down. The result of this heavily altered setup is that the game rewards pure aggression and long-distance fighting much more than in traditional Bomberman games, and the pace of battles is very fast. In the end, Hudson favored the original classic style Bomberman battles, and Bomberman 64: The Second Attack! would be the last time this gridless gameplay style was used in multiplayer.

Team Mode

The objective of this is to destroy the other team's gem. If you die in this mode however, you simply come back to life, and not as a ghost like in Battle mode. The first team to destroy the other teams gem X number of times wins. Up to 4 people can play team mode.

Power-ups:

The Bomb is usually the player's sole method of attack in the Bomberman series. Players have an unlimited amount of bombs at their disposal, and are only hindered by how many they can have on screen at the same time (Bomb-Up) and how large the bombs' explosions are (Fire-Up). We have set the time period of 3.5s for the bomb to explode. In this time period, the player can move to the safest range from the blast point so that he won't die of his own explosion.

Fire-up: The Fire or Fire Up Item is a recurring item in the Bomberman series. It increases the bomb's max blast radius by 1. In grid-based Bomberman games, this means the player's bomb explosion will extend by one tile. In many free-area Bomberman games, this increases the spherical blast radius by a certain amount. Here in this game, when the bomb is exploded, it covers the radius of one block in all four directions and it is increased with each power-ups.

Bomb-up: The Bomb powerup is a recurring item in the Bomberman series. It increases the player's amount of bombs by one. Bomb Capacity is the amount of bombs a player can have on screen at the same time.

Skate power-up: The Skate power-up (also known as Speed Up or Roller-Shoes) is a recurring item in the Bomberman series. It increases the player's speed by 1.

Literature survey:

Since this project is based on the application of object oriented programming concept, different books were suggested to refer by the teachers for the OOP concept. The books like "The secrets of object oriented programming" were referred OOP as reference for the development of program that assisted us to clear the concepts regarding the language and make our program development easier. For implementing the graphics in our project, we have use SFML and for this, different e-books were referred and different video tutorials from the website were viewed.

Existing system:

The original game was published in 1983 and new games have been published at irregular intervals ever since. Today, Bomberman has featured in over 70 different games on numerous platforms (including all Nintendo platforms save for the 3DS and Wii U. One was planned for the 3DS, but was later cancelled), as well as several anime and manga. This franchise is one of the most commercially successful of all time. Being commercially successful, with over 10 million units of games sold, the series "has since become known as the first name in multiplayer games". At the Nintendo Switch Presentation on January 13, 2017, Super Bomberman R was announced as a Nintendo Switch launch title.

Methodology:

This project is based on the "object oriented programming" concept using C++ programming language. Hence, different classes were created with the number of member data and member functions for the smooth running of the program. The concept of “code reusability”, that is, inheritance has been used. The events for each object have been handled by the different member functions so that they form a final outlook working together simultaneously.

C++ has the capability to manage the memory allocation/deallocation on any objects that we've created which can increase the performance of our game. Game is performance critical software that requires 100% usage of the hardware user has, and C++ is only popular language that gives you such abilities:

- High abstraction level - fine Object oriented programming and generic programming
- Very good and deterministic control of the resources you use.
- Ability to optimize special parts to very high level that is almost impossible to achieve with other popular languages.

The strength of C++ when it comes to game development is the ability to exactly layout the data-structures that your software will use. C++ provides the ability to override important performance bottlenecks such as memory allocation. It has the ability to structure and place things exactly where they want in the memory. On top of this it's a flexible programming language that provides a decent development velocity.

This project needed the concept of graphics but since we only had the knowledge of C++ graphics which may not fulfill the required objectives of our project having limited features. For development of the game, we used SFML which uses a 32-bit system and has high resolution which can definitely fulfill the objective of our project. So, we started learning SFML through vavious sites and e-books and understand the basic gaming logic though various gaming sites and forums along with the books reagarding C++ and object oriented programming itself.

Then we had to decide the right compiler and IDE. So, we decided to Code::Blocks-C++ as IDE, which uses GCC compiler. For the graphics and other game related works, we decided to use SFML-2.40 library. After collecting necessary materials, we developed the algorithm of our project and we worked according to it. The coding has been done according to the basic idea of OOP. We have created various classes and related functions and data binding has been used. All the features of C++ like objects, classes, data abstraction and encapsulation, inheritance, polymorphism, dynamic binding, message passing etc. has been used.

SFML is used for various modules: system module for vector and Unicode string classes, portable threading and timer facilities, window module for window and input device management including support for joysticks, OpenGL context management, graphics module for the hardware acceleration of 2D graphics including sprites, polygons and text rendering, audio module for the hardware accelerated spatialised audio playback and recording and network module for the TCP and UDP network sockets, data encapsulation facilities.

The programming methods we've used can as summarized below:

- Analyzing the concept that can be used to develop proper program.
- Discussion on the topic that might be faced onwards.
- Making the project schedule.
- Initial coding for creating logic.
- Coding the program.
- Execution and testing the program.
- Debugging.
- Program Documentation.

The functions used in the program are:

1. MainMenu
    - ♦ class MainMenu - class of main menu
        - ♦ MainMenu(); - MainMenu class constructor, initializes all the required variables for the menu

- int start(); - starts the Main Menu display loop and returns int depending on option chosen in the menu when pressed enter/return, 0 to continue to 2-player game, 1 to exit, called when "enter"ed on exit button or when escape key pressed

2. Engine

- class Engine – class of engine
  - Texture m_BackgroundTexture;
  - Texture m_BackgroundPauseTexture;
  - Texture m_BackgroundPlayer1Wins;
  - Texture m_BackgroundPlayer2Wins;
  - Texture m_FrameTexture;
  - Texture m_BlockTexture;
  - Texture m_SolidBlockTexture;
  - Texture m_PlayerTexture;
  - Texture m_BombTexture;
  - Texture m_PowerTexture5;
  - Texture m_PowerTexture6;
  - Texture m_PowerTexture7;
  - Texture m_ExplosionTexture

  } textures for the background, players,(.png found inside backgrounds and sprites folders)

  - Sprite m_BackgroundSprite;
  - Sprite m_PauseSprite;
  - Sprite m_GameOverSprite[3];
  - Sprite m_FrameSprite;
  - Sprite m_BlockSprite;
  - Sprite m_SolidBlockSprite;
  - Sprite m_PowerSprite5;
  - Sprite m_PowerSprite6;
  - Sprite m_PowerSprite7;

  } normal sprite for Background

  - void input(Time t); - input function called by public member function Engine::start() and does input handling for 3 conditions, game screen, pause screen, and win screen

- ♦ void update(Time dt, Time t); - update function called by public member function Engine::start() and does updating of the sprites according to the condition, runs in game screen only
- ♦ void draw(); - update function called by public member function Engine::start() and does updating of the sprites according to the condition, runs in game screen only
- ♦ void PowerupCheck(Bomber* m_PBomber); - called from within Engine::update() and enables player to pick up PowerUps
- ♦ void resume(); - function for resuming game, sets private bool m_gamePaused to false
- ♦ void pause(); - function for pausing game, sets private bool m_gamePaused to true
- ♦ int gameOver(int loser); - function for finishing game after either player wins or time finishes, the value of int loser corresponds to state and 0-draw, 1-player1 lost, 2-player2 lost
- ♦ void start(); - runs game loop , calls input, update, draw within the loop
- ♦ void updateHUD(); - called from within Engine::update() and updates the HUD

3. Level
   - ♦ void createLevel(std::string file_name, std::vector< std::vector<int> >& Level); - creates the level and reads file_name CSV and stores the values in a vector of (vector of ints) Level
4. Bomber
   - class Bomber
     - ♦ IntRect getCollisionRect(); - gives position vector of the sprite
     - ♦ AnimatedSprite getSprite(); - returns the sprite and called by Engine::draw() to draw on the screen
     - ♦ bool noKeyPressed(); - returns true if no key(movement key) is pressed. Used in update function above to stop
     - ♦ void update(Time dt); - function that updates m_Sprite(animation, position, continue or stop, etc)

- void moveLeft();
- void stopLeft();
- void moveRight();
- void stopRight();
- void moveUp();
- void stopUp();
- void moveDown();
- void stopDown();
- void stop();

} functions to select animation and set and reset the boolean variables defined above like m_LeftPressed

5. Bomb

- class bomb
  - Bomb(Engine* pEngine, Bomber* pBomber, Vector2f position, int range, sf::Time start_time); - constructor for bomb. called from within a bomber class, the bomb object has range "range", and start_time and new bombs are automatically added to Engine::std::vector<Bomb> m_vBombs;
  - AnimatedSprite getSprite(); - returns the animated sprite of the bomb
  - void update(Time dt, Time time); - update the bomb. Takes arguments dt(time since last loop, required for animated class) and time, which is the total time
  - Vector2i getCell(); - returns the position of the bomb in the form of Vector2i(row, column)
  - void blast(); - blasts the bomb. called when time of the bomb is finished and creates an exlopsion object that does the work of blasting, and sets the bool m_blasted to true so that, it doesn't blast again.
  - bool isBlasted(); - returns the private bool m_blasted, that signifies if the bomb has blasted or not

6. Explosion

- class explosion
  - ♦ void explodeDirection(Vector2i startCell, Vector2i unitVector); - explodes in a particular direction depending on the unit vector specified and in the unit vector, one component must be 0 and the other 1 or -1
  - ♦ void destroyBombsInCell(Vector2i currCell); - destroys the bombs in given cell currCell, called by the constructor
  - ♦ bool randomPowerUpDestroy(Vector2i currCell); - if the currCell had a breakable block, randomly choose whether or not to put a Powerup there, and which one to put
  - ♦ void killPlayerinCell(Vector2i currCell); - if the player is in the currCell, this function kills(damages) the Player
  - ♦ Explosion(Engine* pEngine, Vector2i start_cell, int blast_range); - constructor for the Explosion class, does the whole work of the explosion and calls the killPlayerinCell for the start_cell, and calls explodeDirection in 4 directions upto blast_range

7. RCintoCoor

- ♦ sf::Vector2f rcIntoCoor(int r, int c); - changes row r and column c into position vector2f
- ♦ sf::Vector2f coorIntoGood(sf::Vector2f badCoor); - returns the closest good coordinate(a coordinate vector2f that is at the center of a cell) to badCoor, a bad(not-good) coordinate
- ♦ sf::Vector2i coorToRc(sf::Vector2f badCoor); - changes Coordinate vector2f into Vector2i(row, column). Calls coorIntoGood internally, so bad coordinate can also be passed as argument

8. Pausable Timer

- class PausableTimer
  - ♦ PausableTimer(); - constructor for the pausable timer
  - ♦ void Reset(); - Resets the pausable timer
  - ♦ void Start(); - Starts/Resumes the pausable timer

- void Pause(); - Pauses the pausable timer
- float GetElapsedSeconds(); - returns the elapsed time in seconds(type float)

Implementation:

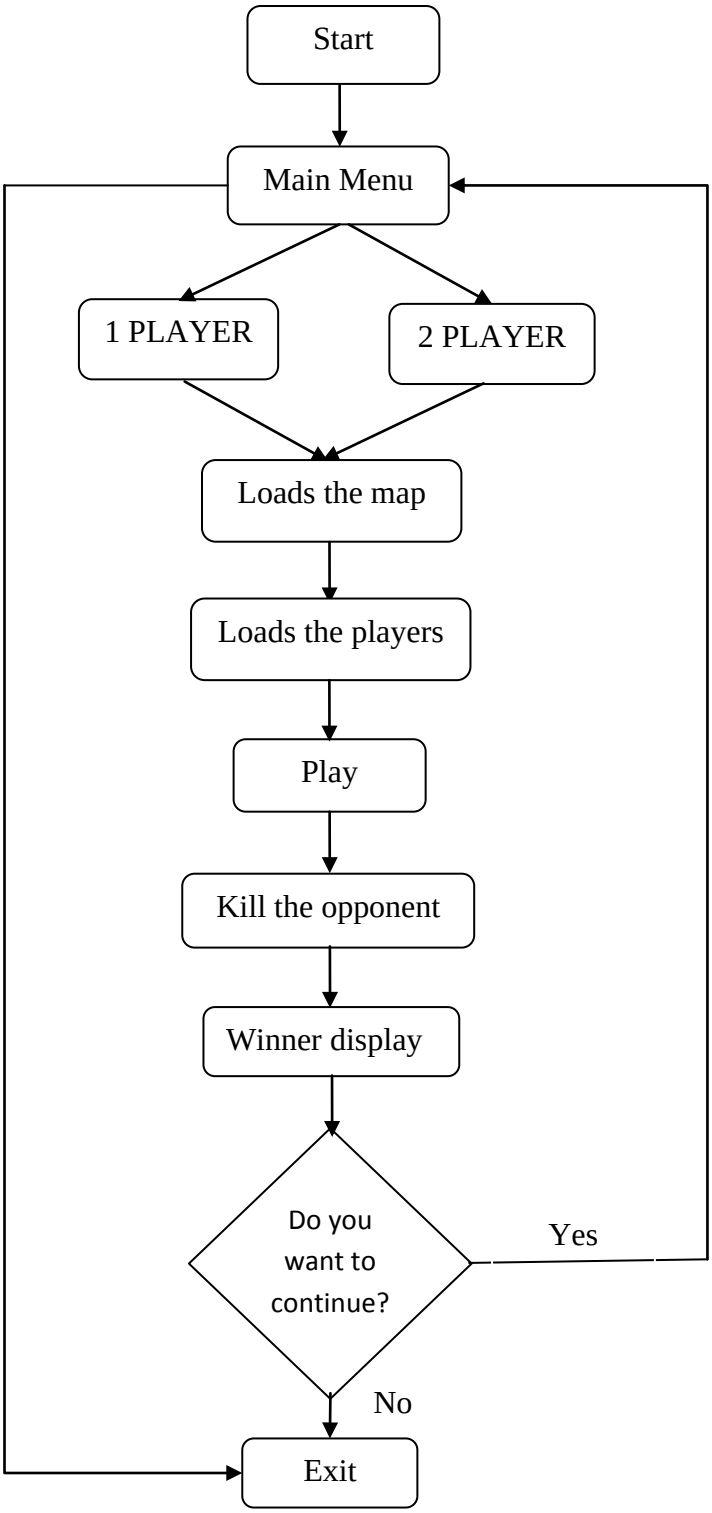
The game starts with the menu that consists of 1 player, 2 player, instructions and exit. There are altogether three maps and a random map appears every time we start the game.

- 1 PLAYER: This allows player to play with computer. In this, if the player touches the opponent, the player dies so the main mission of the player is to eliminate the opponent.
- 2 PLAYER: This allows two users to play the game simultaneously. Similar to one player, the users try to kill each other with the explosion.
- Instruction: This section feeds the user with the information about the keys used for the mobility of the sprites, how to place the bomb, power-ups and other features.
- Exit: This allows user to close the program.

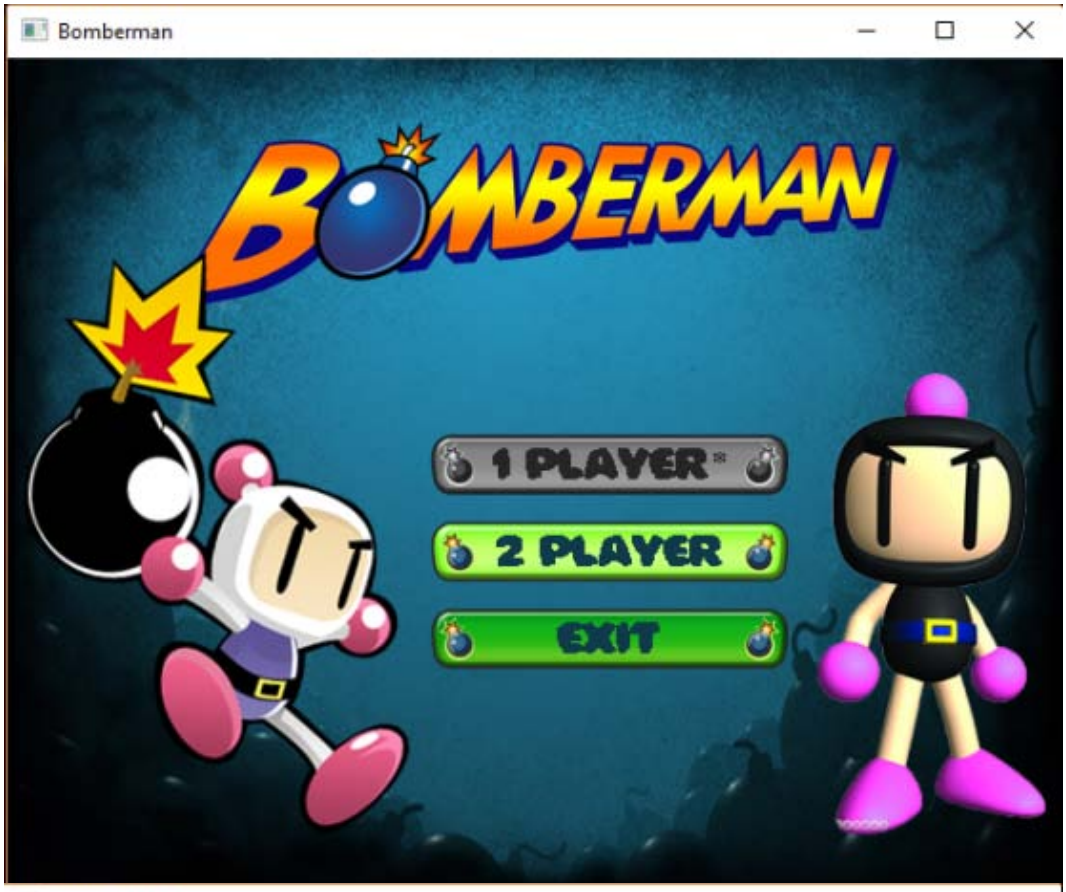
The game will be over within the time limit of 2 minutes if the players couldnot achieve the mission.In overall, the following methods were implemented for the completion of the project, which is regarded as the basic steps for developing the software and listed as follows,

- **Study, research and analysis:** Gathering information about the topic of the project.
- **Proposal submission:** Documenting the information about the project was carried out before starting the coding.
- **Designing of the layout of the project:** Proper positioning of the components required for the project was designed properly.
- **Coding:** Coding of the project was done as mentioned in the proposal.
- **Testing, modification and Documentation:** Project is tested and modified continuously as per requirement to make it more attractive and error free.

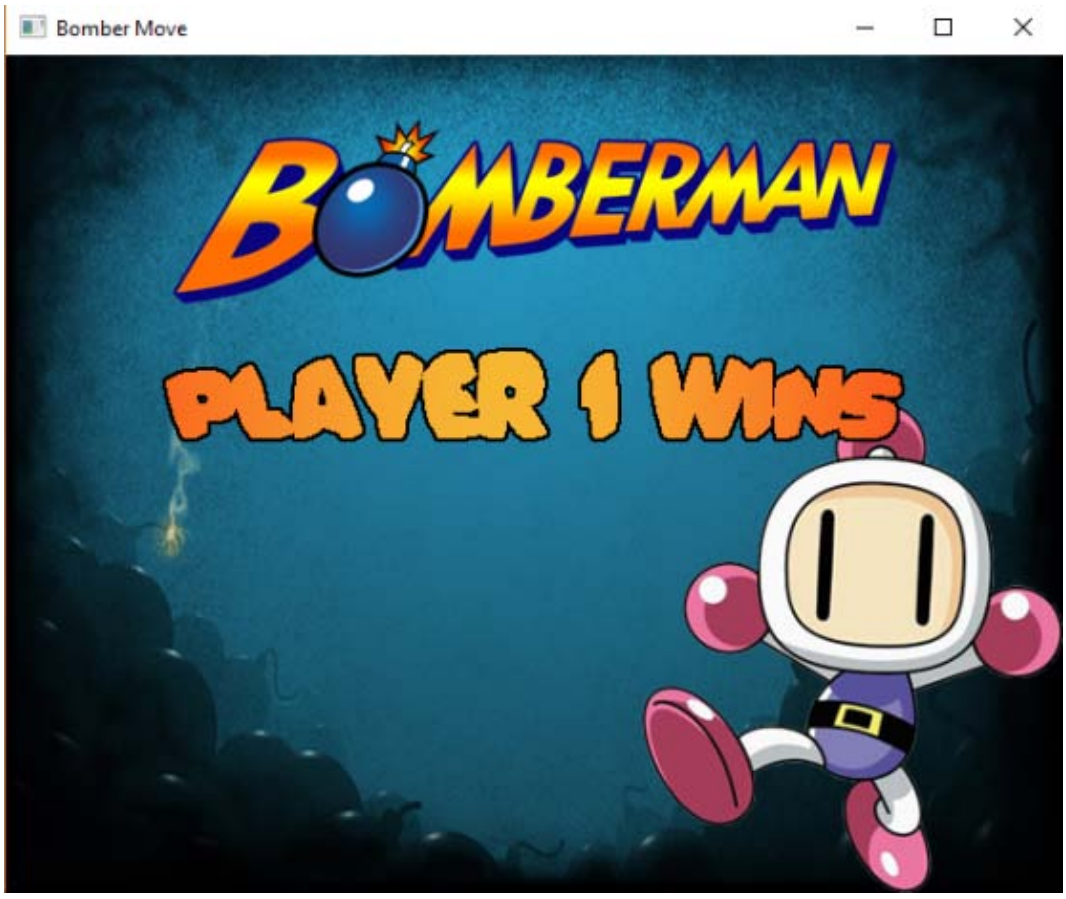
Block-diagram:



Gameplay:

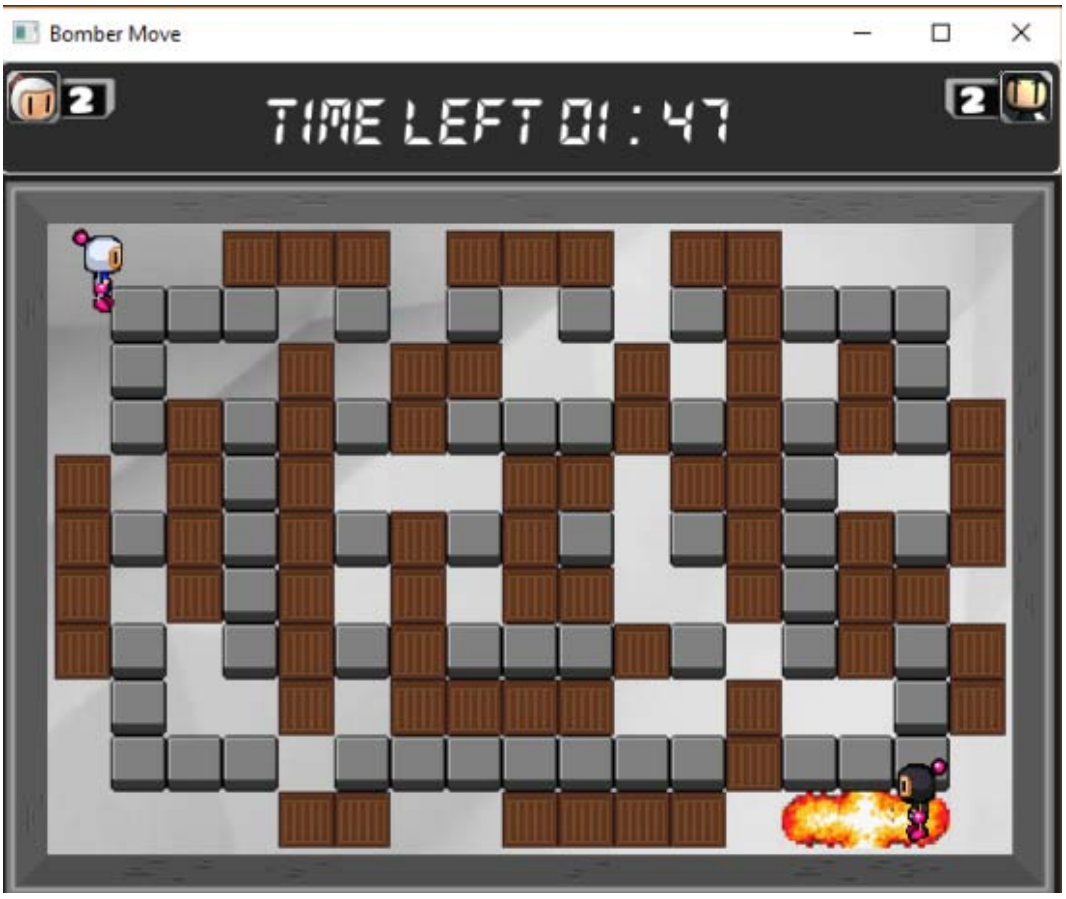

Bomberman
BOMBERMAN
1 PLAYER
2 PLAYER
EXIT


Bomber Move
BOMBERMAN
PLAYER 2 WINS


Bomber Move
BOMBERMAN
PLAYER 1 WINS


Bomber Move
3
TIME LEFT 01:35
3


Bomber Move
2
TIME LEFT 01 : 26
3


Bomber Move
2
TIME LEFT 01 : 47
2

Results:

With the completion of the project, the objective of the project was achieved and the game was designed and completed as per the requirement. The complete game was designed with the necessary features.

The main objective for the development of this game was fulfilled, but still some features in this game could not be completed which might be due to the time constrain, and also, this is the learning phase due to which much time was spent on learning the subject matter as the topic was completely unknown before the starting of project.

From this project, we can take positives as we achieved the co-operation between the team members, learnt to work in a team, backing up each other in need and also the challenges to develop the software by developing this small gaming program.

Problems Faced and Solutions:

This project was almost completed on time with lots of effort. Although the project was completed on time, many problems and errors were faced during the development of the game and the solutions to problems were obtained with the discussion with friends and teachers and the deep study of the codes finally made the error free game. The major problems that occurred during the development of program can be listed as follows:

- As SFML graphics library was completely new for us, we had to go through the e-books and video tutorials related to this library and check and recheck the codes time and again.
- We had problems related to graphics options. We were not known to various graphics function and their workings.
- We had great difficulties in developing proper logic due to lack of proper reference materials.
- We had planned to make this game be played by two players on network. We built a console type system where we were able to pass the data to the computers on network using server-client model. But a runtime error occurred where client can receive and acknowledges the data sent by server. The request sent by the client was not accepted by the server. We didn't know why our code didn't work properly. It followed with the crashing of the server and the graphics card. We tried a lot to solve this error and with limited time with us we opted to drop it from our project.
- We created a vector to store the sprite used in the game which clashed with our animatedsprite file and as a result the program was not responding. So we used two separate objects for the sprites.
- Problems occurred while merging the code from different members like linking the main game with the main menu. The function in one could not be called from the other.
- While working on the explosion of the bombs, we need to include 3 variables for single object so using simple data it was not difficult and when we try to achieve this by making class it was rather confusing and

got mixed up with other classes since it is used only once. Later, it was solved by using the concept of vector of tuples.

Limitations and Future enhancements:

As we've completed the game for entertainment, many more features can still be added to make the game more attractive and more popular among the user so as to make it popular among this competitive market. The following features can be added to our project:

- We had planned to make this game be played by two players on network. We built a console type system where we were able to pass the data to the computers on network using server-client model. But a runtime error occurred where client can receive and acknowledges the data sent by server. The request sent by the client was not accepted by the server. We didn't know why our code didn't work properly. It followed with the crashing of the server and the graphics card. We tried a lot to solve this error and with limited time with us we opted to drop it from our project. By using the application of multithreading with UDP sockets, the game between the two players can be more enhanced.
- This game could be enhanced with different levels and the winner could be displayed with the scored obtained with each progressing so that the gaming will be more interesting and entertaining.
- Different mini games like cart racing, pong, flappy bird, etc can be added in between the game with each progressing so that the user will be entertained with more gaming scope and be more indulged in the gaming.
- While allowing two players in the gameplay, instead of competing with each other we can make a game in which these two players works in the team to compete against the other enemies to complete the given task.
- Tournament type game play can be designed to make the game more interesting using different levels of AI, thus making further competitive among different users.

Conclusion and recommendations:

This project was unquestionably a good way of learning and implementing the way for programming practice. This project leads us to the winding up on the programming practice that for developing software a good judgment and proper analysis of the topic is required at first rather than the coding. The coding is not the initial step for emergent of any program, rather a good planning on the basic framework and making decision on the way of implementing the program is the most. After the coding of the program the system may not be as per our requirement but debugging, if any error, and testing and execution of the program are further more required. After the completion of the system, its management takes, is another most required obsession that is to be handled with great care. Also on using the graphics we might have to load different images for different purpose so proper discussion and proper decision is required as per the situation emerges.

Thus, after the completion of our project, we can conclude proper judgment and implementation of the problems or topic leads to the good programming practice which might lead to have desired output.

References:

- Object Oriented Programming with C++ E Balagurusamy
- Object Oriented Programming with C++ Robert Lafore
- The secrets of Object oriented programming Daya Sagar Baral
- C++ How to program P.J. Dietel & H.M Dietel
- **https://www.sfml-dev.org/tutorials/2.4/**
- **https://www.packtpub.com/game-development/sfml-game-development-example**
- **http://spritedatabase.net**
- **https://www.spriters-resource.com**
- **https://opengameart.org**

Credits:

- Plasma
- Hyptosis
- Zabin
- EddieVanHalen98
- einstein95
- DanielCook
- Silver
- Drshnaps
- Cough-E
- Pingul
- SilverDeoxys563