Road Rampage

Student Name: Joel Medina

Other team members: Simranjeet Singh
Table of Contents

Road Rampage

<table>
<thead>
<tr>
<th>Section</th>
<th>Requirements Workflow 1st Iteration</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.1 Initial Understanding and glossary</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1.2 Initial Business Model</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>1.3 Initial Requirements</td>
<td>13</td>
</tr>
</tbody>
</table>

Section 1: Requirements Workflow 1st Iteration

<table>
<thead>
<tr>
<th>Section</th>
<th>Review of Requirements Workflow by other team member</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.1 Review by team member Simranjeet</td>
<td>14</td>
</tr>
</tbody>
</table>

Section 2: Review of Requirements Workflow by other team member

<table>
<thead>
<tr>
<th>Section</th>
<th>Requirements Workflow 2nd Iteration</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.1 Revised Understanding and glossary</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>3.2 Revised Business Model</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>3.3 Revised Requirements</td>
<td>23</td>
</tr>
</tbody>
</table>

Section 3: Requirements Workflow 2nd Iteration

<table>
<thead>
<tr>
<th>Section</th>
<th>Review of my own Requirements Workflow</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4.1 Self Review</td>
<td>24</td>
</tr>
</tbody>
</table>

Section 4: Review of my own Requirements Workflow

<table>
<thead>
<tr>
<th>Section</th>
<th>Requirements Workflow 3rd Iteration</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5.1 Revised Understanding and glossary</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>5.2 Revised Business Model</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>5.3 Revised Requirements</td>
<td>34</td>
</tr>
</tbody>
</table>

Section 5: Requirements Workflow 3rd Iteration

<table>
<thead>
<tr>
<th>Section</th>
<th>Analysis Workflow 1st Iteration</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6.1 Initial Functional Modeling</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>6.2 Initial Entity Class Modeling</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>6.3 Initial Dynamic Modeling</td>
<td>41</td>
</tr>
</tbody>
</table>

Section 6: Analysis Workflow 1st Iteration

<table>
<thead>
<tr>
<th>Section</th>
<th>Review of Analysis Workflow by other team members</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7.1 Review by team member Joel</td>
<td>43</td>
</tr>
</tbody>
</table>

Section 7: Review of Analysis Workflow by other team members

<table>
<thead>
<tr>
<th>Section</th>
<th>Analysis Workflow 2nd Iteration</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8.1 Revised Functional Modeling</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>8.2 Revised Entity Class Modeling</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>8.3 Revised Dynamic Modeling</td>
<td></td>
</tr>
</tbody>
</table>

Section 8: Analysis Workflow 2nd Iteration

<table>
<thead>
<tr>
<th>Section</th>
<th>Review of my own Analysis Workflow</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9.1 Self Review</td>
<td>50</td>
</tr>
</tbody>
</table>

Section 9: Review of my own Analysis Workflow

<table>
<thead>
<tr>
<th>Section</th>
<th>Analysis Workflow 3rd Iteration</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10.1 Revised Functional Modeling</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>10.2 Revised Entity Class Modeling</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>10.3 Revised Dynamic Modeling</td>
<td>56</td>
</tr>
</tbody>
</table>

Section 10: Analysis Workflow 3rd Iteration

<table>
<thead>
<tr>
<th>Section</th>
<th>Design Workflow 1st Iteration</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11.1 Revised Functional Modeling</td>
<td>58</td>
</tr>
</tbody>
</table>
Section 1: Requirements Workflow 1st Iteration

1.1 Initial Understanding and Glossary

The player’s objective when playing Road Rampage is to avoid traffic and gain points by destroying traffic and other obstacles with a turret mounted on the car controlled by the player. The player will have control of a car on the left hand side which will seem to be moving in the right direction because other cars move towards the player and the background moving as well. The background will consist of a road and foliage on the edges of the road. The player will have a top view of the car and all other objects as if the player was in the sky looking down upon the vehicles and controlling the main vehicle.

Obstacles which must be overcome include traffic and other vehicles which will have the ability to shoot as well. Points will be earned as time goes by and as the player destroys cars are destroyed with the turret’s gunfire. The player will be given a number of lives which will decrease by one every time the player collides with a car or is shot down by an enemy vehicle’s gunfire.

A round of gameplay is depicted as follows. When the game is run the player will be presented with a menu which will consist of three options. The options presented will include Play, High Scores, and Credits and quit. To begin gameplay, the player will have to choose play and then the game will commence. Traffic will begin to appear on the opposite side of the screen relative to the player’s position and the traffic will have to be avoided in order to stay alive. The player will be equipped with a turret mounted on the hood of the car and will be able to control the gunfire of the turret as well as its axis of rotation. The axis of rotation of the turret will be a 180 degree angle, 90 degrees up and 90 degrees down. As gunfire hits other cars, those cars who get hit will be destroyed and be removed from the game.

The player advances in the game by avoiding traffic and gunfire; traffic and enemy cars will increment in speed and difficulty will increase accordingly. The player will have the ability to press pause at any moment in time during the gameplay which will freeze gameplay and bring up a pause menu. The pause menu will consist of two options, either to resume gameplay or return to the main menu. If the player chooses to resume, then gameplay will continue where it was left off. If the player was to choose the main menu option, the player will be returned to the main menu which was initially presented in the first place. One thing to consider about going from gameplay to the main menu is that the player’s score will not be recorded to the scores database. Being destroyed or crashing will decrement the number of lives which the player has. When the player runs out of lives (lives equals zero) then the game will end and the score of the player will be recorded to the table of scores. If the player does not die and continues to advance in the game then score will continue to increase according to time and enemies destroyed until the player himself/herself is destroyed.

If the player chooses the credits option from the main menu then the name of the developer and date of development will be displayed along with the option to return to the main menu.

Glossary
Background: An image of a road which will be rolling to simulate movement of the car.
Car: A vehicle to be controlled by the player
Credits: A display of the developers
Obstacle: Vehicles which include traffic and shooters that the player must overcome.
Menu: A display of options to navigate through different parts of the game.
Shoot: An action made by either a player or an obstacle which fires a projectile to a target
Traffic: Vehicles which will have to be avoided by the player
Turret: A weapon mounted on the vehicle controlled by the player and on some obstacles

1.2 Initial Business Model
Very much business will be done between the user and the software which we have created. A videogames most important aspect is how the user interacts with it, business, and how easily those interactions can be done. Various types of business are done between the player “user” and the game “software.” The business between the user and the game will mainly consist of input buy the user, pressing of the keyboard, and output from the game, results of the player’s inputs. Inputs may include pressing the ‘W’ button which will move the car up as an output; another example may be choosing a selection from a menu which takes the user to that particular place in the game. Whatever the business may be between the user and software, it must be done smoothly and appropriately to optimize the user’s satisfaction and expectations.

Use Cases:

1. Launch Game
2. Choose from Main Menu
3. Move car
4. Shoot
5. Choose from Pause Menu

Case 1: Launch Game

Description: In this case the user decides that he or she would like to play our video game, the main interaction between the user and our software is the launching of the game, whether it is by double clicking it or opening it through some other method. The result of the business being done is that the player will watch the game open and will be presented with the main menu.

Step-by-Step: Not yet applicable

Case 2: Choose from Main Menu
Description: The business being done here between the player and software is that
the software is now presenting an array of selections which the player can choose from.
At this point the player is contemplating on whether he or she would like to play, look at
high scores or the credits. When the player makes a choice, he or she then selects the
option and the results depend on which option is selected.
Step-By-Step: Not yet applicable

Case 3: Move Car

Description: During this use-case the player is given first given the choice to
move the vehicle which they are controlling (the car) up, down, left or right by pressing
the corresponding keys (w, s, a, or d). The choice to make these moves may either be as a
result of actions made by the software’s object(s), or as a step to accomplish a task
towards an object of the software. As a result of a movement made by an obstacle may
include the fact that a vehicle is heading straight for the player and the player makes a
movement to avoid the vehicle. As a step to accomplish a task towards an object, the
player may move up or down to have an obstacle in shooting range and destroy that
obstacle by firing at it.
Step-by-Step: Not yet applicable

Case 4: Shoot

Description: An option to fire the turret which is mounted on the player’s car will also be available to the player. This business between the player and software is to be done as a result of the software’s obstacles approaching and being a danger to the player. After the option to shoot has been taken a projectile will be shot from the turret in the direction of the targeted object. As a result, either the target is destroyed or the projectile misses completely. The player then again has the option to shoot.

Step-by-Step: Not yet applicable.

Case 5: Choose from Pause Menu

Description: During game play the player always had the option to pause by pressing the button "p". When this option is executed the game is paused and the player is then presented with the pause menu which displays two options. The options are either to return to the main menu or resume game play. The player may decide to do this business with the software either because he or she would like to take a brief moment to do something else in reality or because he or she would like to be transferred to the main menu and do business there.

Step-by Step: Not yet applicable.
Use-Case diagrams

Description: The player decides to launch our game and initial parameters are executed.
Step-by-Step:
1. The player decides that he or she would like to play our game
2. The player takes action upon his or her decision and physically launches our videogame software.
3. Initial parameters are executed by the software
4. Player is presented with initial game environment

Description: The main menu is displayed to the player and he or she now has an array of options to choose from.
Step-by-Step:
1. The player is presented the options
2. The player analyzes the options which have been presented
3. The player makes a decision
4. The player acts upon this decision and is taken to the corresponding position in the game based on this decision

Description: Gameplay commences and the player must control the car in order to achieve his or her objectives.

Step-by-Step:
1. Gameplay graphics are presented accordingly
2. The player realizes that he or she must take control of the vehicle
3. The player inputs directional buttons
4. The car moves accordingly
Description: As gameplay persists, the player has the option to shoot the turret, and in a certain instance does so

Step-by-Step:
1. The player sees a chance in which they may make use of the turret
2. The player inputs the shoot button
3. A projectile is shot from the turret and graphics appear accordingly
4. Either the projectile destroys the target, or completely misses

Description: During gameplay the player decides he or she needs to pause the game for any particular reason.

Step-by-Step:
1. The player makes the decision to pause the game
2. The player presses the pause button
3. The pause menu is presented at an instant
4. The player then is given the choice to either resume the game or return to the main menu

Initial Requirements
1. Present well defined graphics along with understanding of environment to the user when the video game is launched.
2. Display self-explanatory options to the player in the main menu along with smooth navigation through the selection process.
3. Allow movement of the car controlled by the player to be smooth and graphics of the vehicle must be well defined.
4. Quick response and precise trajectory when firing the turret.
5. Instant recall to input and easily understood options when game is paused.

Detailed Requirements:

When the user first decides to launch the game, it is critical to initialize the game in a state in which the player can easily understand the environment in which he or she has initially been presented. Throwing the player into a gameplay state in which the player has to immediately interact with the game in order to survive is the wrong way to initialize the user-software business process. A main menu would be the most efficient way to go when first introducing the user to the software. The player will feel comfortable and be able to easily make a choice thanks to self-explanatory options and smooth navigation through the options presented to him or her. Once gameplay has commenced, a critical aspect of user-software interaction is the way in which the car responds to the players inputs. The car must respond quickly and smooth according to inputs made by the player. When firing the turret, not only must the response be instant to input, but the trajectory of the projectile must also be precise and accurate when compared to a normal trajectory. It is critical that the pause menu responds instantly to the player’s input because if it does not, the player may be put in danger by the obstacles of the videogame.
2.1 Review by Team Member Simranjeet Singh

Overview

Overview from Simranjeet

Some descriptions are not clear, revise, make it easy to follow, too jummy (I know)... there is grammar error, need word or remove word used twice. Also, in the use case diagram, connect iteration with "<<include>>" with dotted line, as in the book. Also keep the previous "include" and also show the next with it in next iteration.

Preparation

Preparation from Simranjeet

Everything is detailed and is understandable. All that is mentioned is clear and overall easy to follow.

Inspection:

Everything stated will be fixed.
3.1 Revised Understanding Glossary

The player’s objective when playing Road Rampage is to avoid traffic and gain points by destroying traffic and other obstacles with a turret mounted on the car controlled by the player. The player will have control of a car on the left hand side which will seem to be moving in the right direction because other cars move towards the player and the background moving as well. The background will consist of a road and foliage on the edges of the road. The player will have a top view of the car and all other objects as if the player was in the sky looking down upon the vehicles and controlling the main vehicle.

Obstacles which must be overcome include traffic and other vehicles which will have the ability to shoot as well. Points will be earned as time goes by and as the player destroys cars are destroyed with the turret’s gunfire. The player will be given a number of lives which will decrease by one every time the player collides with a car or is shot down by an enemy vehicle’s gunfire.

A round of gameplay is depicted as follows. When the game is run the player will be presented with a menu which will consist of three options. The options presented will include Play, High Scores, and Credits and quit. To begin gameplay, the player will have to choose play and then the game will commence. Traffic will begin to appear on the opposite side of the screen relative to the player’s position and the traffic will have to be avoided in order to stay alive. The player will be equipped with a turret mounted on the hood of the car and will be able to control the gunfire of the turret as well as its axis of rotation. The axis of rotation of the turret will be a 180 degree angle, 90 degrees up and 90 degrees down. As gunfire hits other cars, those cars who get hit will be destroyed and be removed from the game.

The player advances in the game by avoiding traffic and gunfire; traffic and enemy cars will increment in speed and difficulty will increase accordingly. The player will have the ability to press pause at any moment in time during the gameplay which will freeze gameplay and bring up a pause menu. The pause menu will consist of two options, either to resume gameplay or return to the main menu. If the player chooses to resume, then gameplay will continue where it was left off. If the player was to choose the main menu option, the player will be returned to the main menu which was initially presented in the first place. One thing to consider about going from gameplay to the main menu is that the player’s score will not be recorded to the scores database. Being destroyed or crashing will decrement the number of lives which the player has. When the player runs out of lives (lives equals zero) then the game will end and the score of the player will be recorded to the table of scores. If the player does not die and continues to advance in the game then score will continue to increase according to time and enemies destroyed until the player himself/herself is destroyed.

If the player chooses the credits option from the main menu then the name of the developer and date of development will be displayed along with the option to return to the main menu.

Glossary
Background: An image of a road which will be rolling to simulate movement of the car.
Car: A vehicle to be controlled by the player
Credits: A display of the developers
Obstacle: Vehicles which include traffic and shooters that the player must overcome.
Menu: A display of options to navigate through different parts of the game.
Shoot: An action made by either a player or an obstacle which fires a projectile to a target
Traffic: Vehicles which will have to be avoided by the player
Turret: A weapon mounted on the vehicle controlled by the player and on some obstacles

3.2 Revised Business Model
Very much business will be done between the user and the software which we have created. A videogame's most important aspect is how the user interacts with it, business, and how easily those interactions can be done. Various types of business are done between the player “user” and the game “software.” The business between the user and the game will mainly consist of input by the user, pressing of the keyboard, and output from the game, results of the player’s inputs. Inputs may include pressing the ‘W’ button which will move the car up as an output; another example may be choosing a selection from a menu which takes the user to that particular place in the game. Whatever the business may be between the user and software, it must be done smoothly and appropriately to optimize the user’s satisfaction and expectations.

Use Cases:

1. Launch Game
2. Choose from Main Menu
3. Move car
4. Shoot
5. Choose from Pause Menu

Case 1: Launch Game

Description: In this case the user decides that he or she would like to play our video game, the main interaction between the user and our software is the launching of the game, whether it is by double clicking it or opening it through some other method. The result of the business being done is that the player will watch the game open and will be presented with the main menu.

Step-by-Step: Not yet applicable

Case 2: Choose from Main Menu
Description: The business being done here between the player and software is that the software is now presenting an array of selections which the player can choose from. At this point the player is contemplating on whether he or she would like to play, look at high scores or the credits. When the player makes a choice, he or she then selects the option and the results depend on which option is selected.

Step-By-Step: Not yet applicable

Case 3: Move Car

Description: During this use-case the player is given first given the choice to move the vehicle which they are controlling (the car) up, down, left or right by pressing the corresponding keys (w, s, a, or d). The choice to make these moves may either be as a result of actions made by the software’s object(s), or as a step to accomplish a task towards an object of the software. As a result of a movement made by an obstacle may include the fact that a vehicle is heading straight for the player and the player makes a movement to avoid the vehicle. As a step to accomplish a task towards an object, the player may move up or down to have an obstacle in shooting range and destroy that obstacle by firing at it.

Step-by-Step: Not yet applicable
Case 4: Shoot

Description: An option to fire the turret which is mounted on the player’s car will also be available to the player. This business between the player and software is to be done as a result of the software’s obstacles approaching and being a danger to the player. After the option to shoot has been taken a projectile will be shot from the turret in the direction of the targeted object. As a result, either the target is destroyed or the projectile misses completely. The player then again has the option to shoot.

Step-by-Step: Not yet applicable.

Case 5: Choose from Pause Menu

Description: During game play the player always had the option to pause by pressing the button ‘p’. When this option is executed the game is paused and the player is then presented with the pause menu which displays two options. The options are either to return to the main menu or resume game play. The player may decide to do this business with the software either because he or she would like to take a brief moment to do something else in reality or because he or she would like to be transferred to the main menu and do business there.

Step-by-Step: Not yet applicable.

Use-Case diagrams
Description: The player decides to launch our game and initial parameters are executed.

Step-by-Step:
1. The player decides that he or she would like to play our game
2. The player takes action upon his or her decision and physically launches our videogame software.
3. Initial parameters are executed by the software
4. Player is presented with initial game environment.

Description: The main menu is displayed to the player and he or she now has an array of options to choose from.

Step-by-Step:
1. The player is presented the options
2. The player analyzes the options which have been presented
3. The player makes a decision
4. The player acts upon this decision and is taken to the corresponding position in the game based on this decision

Description: Gameplay commences and the player must control the car in order to achieve his or her objectives.

Step-by-Step:
1. Gameplay graphics are presented accordingly
2. The player realizes that he or she must take control of the vehicle
3. The player inputs directional buttons
4. The car moves accordingly

Description: As gameplay persists, the player has the option to shoot the turret.
and in a certain instance does so

Step-by-Step:
1. The player sees a chance in which they may make use of the turret
2. The player inputs the shoot button
3. A projectile is shot from the turret and graphics appear accordingly
4. Either the projectile destroys the target, or completely misses

Description: During gameplay the player decides he or she needs to pause the
game for any particular reason.
Step-by-Step:
1. The player makes the decision to pause the game
2. The player presses the pause button
3. The pause menu is presented at an instant
4. The player then is given the choice to either resume the game or
   return to the main menu

3.3 Revised Requirements
1. Present well defined graphics along with understanding of environment to the user when the video game is launched.
2. Display self-explanatory options to the player in the main menu along with smooth navigation through the selection process.
3. Allow movement of the car controlled by the player to be smooth and graphics of the vehicle must be well defined.
4. Quick response and precise trajectory when firing the turret.
5. Instant recall to input and easily understood options when game is paused.

Detailed Requirements:
When the user first decides to launch the game, it is critical to initialize the game in a state in which the player can easily understand the environment in which he or she has initially been presented. Throwing the player into a gameplay state in which the player has to immediately interact with the game in order to survive is the wrong way to initialize the user-software business process. A main menu would be the most efficient way to go when first introducing the user to the software. The player will feel comfortable and be able to easily make a choice thanks to self-explanatory options and smooth navigation through the options presented to him or her. Once gameplay has commenced, a critical aspect of user-software interaction is the way in which the car responds to the players inputs. The car must respond quickly and smooth according to inputs made by the player. When firing the turret, not only must the response be instant to input, but the trajectory of the projectile must also be precise and accurate when compared to a normal trajectory. It is critical that the pause menu responds instantly to the player’s input because if it does not, the player may be put in danger by the obstacles of the videogame.

Section 4: Review of Own requirements Workflow
After reviewing my own work various times, and reading and understanding exactly what is needed in the requirements workflow and how it needs to be done I have come to find faults which will be fixed in the third iteration of the requirements workflow.

Overview:
- The understanding of the software must be as clear and precise as possible.
- The glossary seems too short.
- The use cases must be at optimum precision for easy understanding
- The final requirements must have the iterated use case diagram with a detailed description of each use case.

Preparation:
- The understanding of the software will be reviewed various times and edited for any errors in grammar or explanation, also an update to the understanding will be made.
- All terms to be used in the software development will be covered and a very detailed definition will be given.
- The use cases will be edited for any errors in explanation and grammar for optimum understanding.
- The final requirements will have the iterated use case diagram with a very detailed description and in good form.

Addition Comments:
- An explanation to the purpose for everything which is being done will also be included at the beginning of every section for better understanding to anyone not familiar with the software engineering process.

Section 5: Requirements workflow 3rd Iteration
5.1 Revised Understanding and Glossary

The following sets of paragraphs will explain what the software to be created will consist of and every aspect of the software from the user’s perspective. This is done to give the reader of this article an explanation of what the software will be and perhaps some insight on what the developer will be up against when designing and creating the software.

Understanding: An Explanation of Road Rampage and what it consists of

The player’s objective when playing Road Rampage is to avoid traffic and gain points by destroying traffic and other obstacles. The player will be able to destroy the obstacles with a turret mounted on the car. The player will have control of a car on the right hand side which will seem to be moving in the left direction because other cars move towards the player and the background moving as well. The background will consist of a road and foliage on the edges of the road. The player will have a top view of the car and all other objects as if the player was in the sky looking down upon the vehicles and controlling the main vehicle.

Obstacles which must be overcome include traffic and other vehicles which will have the ability to shoot as well. Points will be earned as time goes by and as the player destroys cars with the turret’s gunfire. The player will be given a number of lives which will decrease by one every time the player collides with a car or is shot down by an enemy vehicle’s gunfire.

A round of game play is depicted as follows. When the game is initiated the player will be presented with a menu which will consist of three options. The options presented will include Play and quit. To begin game play, the player will have to choose play and then the game will commence. Traffic will begin to appear on the opposite side of the screen relative to the player’s position and the traffic will have to be avoided in order to stay alive. The player will be equipped with a turret mounted on the top of the car and will be able to control the gunfire of the turret as well as its axis of rotation. The axis of rotation of the turret will be a 90 degree angle, 45 degrees up and 45 degrees down. As gunfire hits other cars, those cars who get hit will be destroyed and be removed from the game.

The player advances in the game by avoiding traffic and gunfire; traffic and enemy cars will increment in speed and difficulty will increase accordingly. The player will have the ability to press pause at any moment in time during the game play which will freeze game play and bring up a pause menu. The pause menu will consist of two options, either to resume game play or return to the main menu. If the player chooses to resume, then game play will continue where it was left off. If the player was to choose the main menu option, the player will be returned to the main menu which was initially presented. Being destroyed or crashing will decrement the number of lives which the player has. When the player runs out of lives (when lives equals zero) then the game will end and the player will be returned to the main menu. If the player does not die and continues to advance in the game then score will continue to increase according to time and enemies destroyed until the player himself or herself is destroyed.
Glossary

The following terms are terms belonging to the software which will be used throughout all workflows refer to certain things. This glossary was created in order to clear up any misunderstanding of terms which may spring up during the complete software engineering processes.

**Background:** An image of a road which will be rolling to simulate movement of the car.
**C++:** The programming language which will be used to create the software
**Car:** The vehicle to be controlled by the player
**Credits:** A display of the developers
**Obstacle:** Vehicles which include traffic and shooters that the player must overcome.
**OpenGL:** Graphics library API which will be used to create graphics in the game
**Menu:** A display of options to navigate through different parts of the game.
**Player:** The person who will be using the software product after it is finished
**Projectile:** A particle which is shot or launched, in this case a bullet shot from the turret
**Shoot:** An action made by either a player or an obstacle which fires a projectile to a target
**Traffic:** Vehicles which will have to be avoided by the player
**Turret:** A weapon mounted on the vehicle controlled by the player and on some obstacles
**Visual C++ 6.0:** The integrated development environment which will be used to create the software product

5.2 Revised Business Model
Very much business will be done between the user and the software. A videogame’s most important aspect is how the user interacts with it (business) and how easily those interactions can be done. Various types of business are done between the player “user” and the game “software.” The business between the user and the game will mainly consist of input by the user, pressing of the keyboard, and output from the game, results of the player’s inputs in the form of graphics being displayed on the screen. Inputs may include pressing the ‘W’ button which will move the car up as an output; another example may be choosing a selection from a menu which takes the user to that particular place in the game. Whatever the business may be between the user and software, it must be done smoothly and appropriately to optimize the user’s satisfaction and expectations. The following diagrams are called use-cases and are made to depict business which will be done between the user and the software.

Use Cases:

1. Launch Game
2. Choose from Main Menu
3. Move car
4. Shoot
5. Choose from Pause Menu

Use Case 1: Launch Game

Description: In this case the user decides that he or she would like to play our video game, the main interaction between the user and our software is the launching of the game, whether it is by double clicking it or opening it through some other method. The result of the business being done is that the player will watch the game open and will be presented with the main menu.

Step-by-Step: Not yet applicable
Use Case 2: Choose from Main Menu

Description: The business being done here between the player and software is that the software is now presenting an array of selections which the player can choose from. At this point the player is contemplating on whether he or she would like to play, look at high scores or the credits. When the player makes a choice, he or she then selects the option and the results depend on which option is selected.

Step-By-Step: Not yet applicable

Use Case 3: Move Car

Description: During this use-case the player is given first given the choice to move the vehicle which they are controlling (the car) up, down, left or right by pressing the corresponding keys (w, s, a, or d). The choice to make these moves may either be as a result of actions made by the software’s object(s), or as a step to accomplish a task towards an object of the software. As a result of a movement made by an obstacle may include the fact that a vehicle is heading straight for the player and the player makes a movement to avoid the vehicle. As a step to accomplish a task towards an object, the player may move up or down to have an obstacle in shooting range and destroy that
obstacle by firing at it.
Step-by-Step: Not yet applicable

Use Case 4: Shoot

Description: An option to fire the turret which is mounted on the player’s car will also be available to the player. This business between the player and software is to be done as a result of the software’s obstacles approaching and being a danger to the player. After the option to shoot has been taken a projectile will be shot from the turret in the direction of the targeted object. As a result, either the target is destroyed or the projectile misses completely. The player then again has the option to shoot.
Step-by-Step: Not yet applicable.

Use Case 5: Choose from Pause Menu

Description: During game play the player always had the option to pause by pressing the button ‘p’. When this option is executed the game is paused and the player is then presented with the pause menu which displays two options. The options are either to return to the main menu or resume game play. The player may decide to do this business with the software either because he or she would like to take a brief moment to do something else in reality or because he or she would like to be transferred to the main menu and do business there.
Step-by-Step: Not yet applicable.
Use-Case diagrams

Description: The player decides to launch our game and initial parameters are executed.

Step-by-Step:

1. The player decides that he or she would like to play our game
2. The player takes action upon his or her decision and physically launches our videogame software.
3. Initial parameters are executed by the software
4. Player is presented with initial game environment.
Description: The main menu is displayed to the player and he or she now has an array of options to choose from.

Step-by-Step:
1. The player is presented the options
2. The player analyzes the options which have been presented
3. The player makes a decision
4. The player acts upon this decision and is taken to the corresponding position in the game based on this decision

Description: Gameplay commences and the player must control the car in order to achieve his or her objectives.

Step-by-Step:
1. Gameplay graphics are presented accordingly
2. The player realizes that he or she must take control of the vehicle
3. The player inputs directional buttons
4. The car moves accordingly

Description: As gameplay persists, the player has the option to shoot the turret, and in a certain instance does so

Step-by-Step:
1. The player sees a chance in which they may make use of the turret
2. The player inputs the shoot button
3. A projectile is shot from the turret and graphics appear accordingly
4. Either the projectile destroys the target, or completely misses

Description: During gameplay the player decides he or she needs to pause the game for any particular reason.
Step-by-Step:
1. The player makes the decision to pause the game
2. The player presses the pause button
3. The pause menu is presented at an instant
   5. The player then is given the choice to either resume the game or return to the main menu
5.3 Revised Requirements

1. Present well defined graphics along with understanding of environment to the user when the video game is launched.
2. Display self-explanatory options to the player in the main menu along with smooth navigation through the selection process.
3. Allow movement of the car controlled by the player to be smooth and graphics of the vehicle must be well defined.
4. Quick response and precise trajectory when firing the turret.
5. Instant recall to input and easily understood options when game is paused.

Detailed Requirements:

When the user first decides to launch the game, it is critical to initialize the game in a state in which the player can easily understand the environment in which he or she has initially been presented. Throwing the player into a game play state in which the player has to immediately interact with the game in order to survive is the wrong way to initialize the user-software business process. A main menu would be the most efficient way to go when first introducing the user to the software. The player will feel comfortable and be able to easily make a choice thanks to self-explanatory options and smooth navigation through the options presented to him or her. Once game play has commenced, a critical aspect of user-software interaction is the way in which the car responds to the players inputs. The car must respond quickly and smooth according to inputs made by the player. When firing the turret, not only must the response be instant to input, but the trajectory of the projectile must also be precise and accurate when compared to a normal trajectory. It is critical that the pause menu responds instantly to the player’s input because if it does not, the player may be put in danger by the obstacles of the videogame.

The following use-case diagram will be accompanied by descriptions to every use-case in the diagram. The descriptions will also include requirements in order for business done in those use-cases to occur smoothly.
Launch Game:
Description: The First business to be done between the user and software, as stated before, it is very important to introduce the player to the video game in a smooth manner by presenting the player to the main menu.

Choose From Main Menu:
Description: A well formatted list of initial options will be presented to the player, after making a final decision the player will press the proper inputs and make a choice which will take him or her into another part of the game. It is important to have the menu in a good format and to make the navigation between the options very smooth.

Move Car:
Description: This is perhaps the most important business which will be done between the user and software. During game play the player has the option to navigate throughout the road using a car. Pressing either up, down, left or right will take the player in the direction which they please. The controls must be smooth and must respond immediately to inputs. Multiple input handling must be integrated.

Shoot:
Description: During game play the player will have the option to shoot the turret which will be mounted on the car. Again inputs must be immediate in order to meet the player’s needs in order to survive throughout the game. Graphics must also appear immediately, and any calculations which must be done must be done with accuracy and precision.

Pause:
Description: During game play the player always has the option to pause the game. This action must be followed by immediate stopping of the game and showing of the pause menu. Time is very important when doing this business because if the pause does not occur immediately after the input has been made, then the player could lose without it being their fault at all.
Section 6: Analysis Workflow 1\textsuperscript{st} Iteration

6.1 Initial Functional Modeling

Normal Scenario:

1. The user decides he or she would like to play our videogame “Road Rampage”
2. The player is then presented with the initial main menu
3. The player analyzes the options which have been presented to him or her
4. The player places the cursor on the “Play Game” selection
5. The player presses the selection button “Enter”
6. Game play commences
7. A few second are given to allow the player to analyze his or her position in the game
8. Obstacles consisting of other vehicles begin to appear on the opposite side of the screen and approach the car controlled by the player
9. The player realizes that he or she must move his or her current position in order to avoid traffic
10. The player presses a directional button (Either w, s, a or d)
11. The car controlled by the player moves accordingly
12. Traffic is avoided
13. The player then notices that traffic is becoming unavoidable
14. The player aims with the turret at the unavoidable vehicle
15. The player presses the shoot button “Spacebar”
16. A projectile is shot from the turret
17. The projectile hit’s the enemy vehicle and is avoided
18. Another enemy vehicle begins to shoot projectiles in the direction of the player
19. The player cannot respond fast enough to avoid the projectile
20. The car controlled by the player is hit by the projectile
21. The car controlled by the player is destroyed
22. The score gained by the player is then displayed on the screen
23. The score is saved on the high score chart
24. The player is brought back to the main menu
25. The player places the cursor on the quit option
26. The player presses the selection button “enter”
27. The game exits

Exception Scenario:

1. The user decides he or she would like to play our videogame “Road Rampage”
2. The player is then presented with the initial main menu
3. The player analyzes the options which have been presented to him or her
4. The player places the cursor on the “Play Game” selection
5. The player presses the selection button “Enter”
6. Game play commences
7. A few second are given to allow the player to analyze his or her position in the game
8. Obstacles consisting of other vehicles begin to appear on the opposite side of the screen and approach the car controlled by the player
9. The player realizes that he or she must move his or her current position in order to avoid traffic
10. The player presses a directional button (Either w, s, a or d)
11. The car controlled by the player moves accordingly
12. Traffic is avoided
13. The player then decided that he or she does not want to play anymore
14. The player presses the pause button “p”
15. Game play is paused
16. The pause menu is presented
17. The player analyzes the pause menu
18. The player places the cursor on the “return to main menu” option
19. The player presses the selection button “enter”
20. The player is taken back to the main menu
21. The player places the cursor on the “Quit” option
22. The player presses the selection button “enter”
23. The game exits
6.2 Initial Entity Class Modeling

Game description in one paragraph.

Road rampage is a game in which firstly one is presented with a menu to select whether one would like to play, check high scores, or look at credits. After selecting play, one must control a car to avoid traffic and other obstacles. A turret will be attached to the hood of the car which allows the player to launch a projectile and destroy the obstacles. Points are gained as time passes and as enemies are destroyed.

Noun extraction

**Road rampage** is a **game** in which firstly **one** is presented with a **menu** to select whether **one** would like to play, check high **scores**, or look at **credits**. After selecting **play**, one must control a **car** to avoid **collision** with **traffic** and other **obstacles** like **shooters**. A **turret** will be attached to the **hood** of the **car** which allows the **player** to launch a **projectile** and destroy the **obstacles**. **Points** are gained as **time** passes and as **enemies** are destroyed.

Entity classes:
- Menu, scores, credits, car, collision, traffic, turret, projectile, time

The projectile class communicates with the collision controller class, because when a projectile comes in contact with a shooter or traffic, it is also treated as a collision.
**Boundary Class Extraction**

The boundary class which corresponds to the videogame will be a set of modules which will handle interaction and communication between the user and the software. Modules which correspond to the boundary class include a module which will detect when a keystroke is made, and will then make the corresponding changes to the game. Another module belonging to the boundary class will handle outputs to the screen and the graphics which will be loaded.

**Control Class Modeling**

The control class in the videogame is the collision controller because it is where entities will meet and this controller class will handle any interaction between the entities. What the collision controller class does is it detects any collisions made between the car class and the traffic, shooter or projectile classes and collisions made between the projectile class and traffic or shooter classes.
6.3 Initial Dynamic Modeling

Through the loops of the game there will be constant checks for collisions between entities all controlled by the collision module. In the event of a collision, corresponding graphics will be updated. If a collision between the car and another module occurs then the loop will terminate.

Communication Model

Communication within the modules is done primarily by the collision controller. Positions of entities are extracted and distances are calculated. If a distance less than a certain value depending on the entity, then collision is made.
Section 7: Review of Analysis Workflow by other Team Member

7.1 Review by Simranjeet Singh

Overview

Overview from Team member A
In first scenario shouldn't it be that the player is hit and he loses a life, and if this happens for certain times then player loses the game and then the score is displayed. When diagram of entity classes is described it is a bit unclear to understand, revision will help. Make communication model clearer, easy to the eyes and to understand.

Preparation

Preparation from Team member A
Everything is detailed and is understandable.
Section 8: Analysis Workflow 2\textsuperscript{nd} Iteration

8.1 Revised Functional Modeling

Normal Scenario:

1. The user decides he or she would like to play our videogame “Road Rampage”
2. The player is then presented with the initial main menu
3. The player analyzes the options which have been presented to him or her
4. The player places the cursor on the “Play Game” selection
5. The player presses the selection button “Enter”
6. Game play commences
7. A few seconds are given to allow the player to analyze his or her position in the game
8. Obstacles consisting of other vehicles begin to appear on the opposite side of the screen and approach the car controlled by the player
9. The player realizes that he or she must move his or her current position in order to avoid traffic
10. The player presses a directional button (Either w, s, a or d)
11. The car controlled by the player moves accordingly
12. Traffic is avoided
13. The player then notices that traffic is becoming unavoidable
14. The player aims with the turret at the unavoidable vehicle
15. The player presses the shoot button “Spacebar”
16. A projectile is shot from the turret
17. The projectile hits the enemy vehicle and is avoided
18. Another enemy vehicle begins to shoot projectiles in the direction of the player
19. The player cannot respond fast enough to avoid the projectile
20. The car controlled by the player is hit by the projectile
21. The car controlled by the player is destroyed
22. The score gained by the player is then displayed on the screen
23. The score is saved on the high score chart
24. The player is brought back to the main menu
25. The player places the cursor on the quit option
26. The player presses the selection button “enter”
27. The game exits

Exception Scenario:

1. The user decides he or she would like to play our videogame “Road Rampage”
2. The player is then presented with the initial main menu
3. The player analyzes the options which have been presented to him or her
4. The player places the cursor on the “Play Game” selection
5. The player presses the selection button “Enter”
6. Game play commences
7. A few seconds are given to allow the player to analyze his or her position in the game
8. Obstacles consisting of other vehicles begin to appear on the opposite side of the screen and approach the car controlled by the player
9. The player realizes that he or she must move his or her current position in order to avoid traffic
10. The player presses a directional button (Either w, s, a or d)
11. The car controlled by the player moves accordingly
12. Traffic is avoided
13. The player then decided that he or she does not want to play anymore
14. The player presses the pause button “p”
15. Game play is paused
16. The pause menu is presented
17. The player analyzes the pause menu
18. The player places the cursor on the “return to main menu” option
19. The player presses the selection button “enter”
20. The player is taken back to the main menu
21. The player places the cursor on the “Quit” option
22. The player presses the selection button “enter”
23. The game exits
8.2 Revised Entity Class Modeling

Game description in one paragraph.

Road rampage is a game in which firstly one is presented with a menu to select whether one would like to play, check high scores, or look at credits. After selecting play, one must control a car to avoid traffic and other obstacles. A turret will be attached to the hood of the car which allows the player to launch a projectile and destroy the obstacles. Points are gained as time passes and as enemies are destroyed.

Noun extraction

Road rampage is a game in which firstly one is presented with a menu to select whether one would like to play, check high scores, or look at credits. After selecting play, one must control a car to avoid collision with traffic and other obstacles like shooters. A turret will be attached to the hood of the car which allows the player to launch a projectile and destroy the obstacles. Points are gained as time passes and as enemies are destroyed.

Entity classes:
Car, Collision, Traffic, Turret, Projectile, Shooters

The projectile class communicates with the collision controller class, because when a projectile comes in contact with a shooter or traffic, it is also treated as a collision.

Boundary Class Extraction

The boundary class which corresponds to the videogame will be a set of modules
which will handle interaction and communication between the user and the software. Modules which correspond to the boundary class include a module which will detect when a keystroke is made, and will then make the corresponding changes to the game. Another module belonging to the boundary class will handle outputs to the screen and the graphics which will be loaded.

Control Class Modeling

The control class in the videogame is the collision controller because it is where entities will meet and this controller class will handle any interaction between the entities. What the collision controller class does is it detects any collisions made between the car class and the traffic, shooter or projectile classes and collisions made between the projectile class and traffic or shooter classes. The classes which must be checked for collision will send their coordinates to the collision controller, then the collision controller will decide whether a collision is made or not.
8.3 Revised Dynamic Modeling

State Chart

Through the loops of the game there will be constant checks for collisions between entities all controlled by the collision module. In the event of a collision, corresponding graphics will be updated. If a collision between the car and another module occurs then the loop will terminate. Also user’s inputs will be constantly checked for, when a button is pressed then the state at which the game is will change depending on what was inputted.

Communication Model

Communication within the modules is done primarily by the collision controller. Positions of entities are extracted and distances are calculated. If a distance less than a certain value depending on the entity, then collision is made. When a collision is made the Graphics are updated to show to the user that a collision has occurred.
Section 9: Review of Own Analysis Workflow

9.1 Self Review

Overview:
- The Functional Modeling will be edited and modified.
- The Entity Class modeling will also be edited and diagrams need to be updated.
- The Dynamic Modeling will need to be updated with the iterated charts.

Preparation:
- The Functional Modeling will be edited and modified if necessary to update any changes, and optimize readability.
- The Entity Class modeling will also be edited and diagrams will be updated to show a better depiction of the classes’ positions and which classes communicate.
- The Dynamic Modeling will be update with the iterated charts which will show exactly how game states will change and modules communicate.
Section 10: Analysis Workflow 3\textsuperscript{rd} Iteration

10.1 Revised Functional Modeling

The following diagram and descriptions will depict a complete scenario of the Road Rampage game. A scenario is the instantiation of the use-cases which the game will have. All of the business which had been depicted by the use-cases will be included in the following scenarios.

Normal Scenario:

1. The user decides he or she would like to play our videogame “Road Rampage”
2. The player is then presented with the initial main menu
3. The player analyzes the options which have been presented to him or her
4. The player places the cursor on the “Play Game” selection
5. The player presses the selection button “Enter”
6. Game play commences
7. A few second are given to allow the player to analyze his or her position in the game
8. Obstacles consisting of other vehicles begin to appear on the opposite side of the screen and approach the car controlled by the player
9. The player realizes that he or she must move his or her current position in order to avoid traffic
10. The player presses a directional button (Either w, s, a or d)
11. The car controlled by the player moves accordingly
12. Traffic is avoided
13. The player then notices that traffic is becoming unavoidable
14. The player aims with the turret at the unavoidable vehicle
15. The player presses the shoot button “Spacebar”
16. A projectile is shot from the turret
17. The projectile hits the enemy vehicle and is avoided
18. Another enemy vehicle begins to shoot projectiles in the direction of the player
19. The player cannot respond fast enough to avoid the projectile
20. The car controlled by the player is hit by the projectile
21. The car controlled by the player is destroyed
22. The score gained by the player is then displayed on the screen
23. The score is saved on the high score chart
24. The player is brought back to the main menu
25. The player places the cursor on the quit option
26. The player presses the selection button “enter”
27. The game exits

Exception Scenario:

1. The user decides he or she would like to play our videogame “Road Rampage”
2. The player is then presented with the initial main menu
3. The player analyzes the options which have been presented to him or her
4. The player places the cursor on the “Play Game” selection
5. The player presses the selection button “Enter”
6. Game play commences
7. A few seconds are given to allow the player to analyze his or her position in the game
8. Obstacles consisting of other vehicles begin to appear on the opposite side of the screen and approach the car controlled by the player
9. The player realizes that he or she must move his or her current position in order to avoid traffic
10. The player presses a directional button (Either w, s, a or d)
11. The car controlled by the player moves accordingly
12. Traffic is avoided
13. The player then decides that he or she does not want to play anymore
14. The player presses the pause button “p”
15. Game play is paused
16. The pause menu is presented
17. The player analyzes the pause menu
18. The player places the cursor on the “return to main menu” option
19. The player presses the selection button “enter”
20. The player is taken back to the main menu
21. The player places the cursor on the “Quit” option
22. The player presses the selection button “enter”
23. The game exits
10.2 Revised Entity Class Modeling

In the Road Rampage game there will be several entities which will most likely end up as being classes in our design and eventually in the code. To point out the entities which are most likely to end up being classes the game will be described in a paragraph and the nouns in the paragraph will be extracted and taken into consideration when choosing what entities will be represented by classes. The following paragraph describes the Road Rampage game.

Road rampage is a game in which firstly one is presented with a menu to select whether one would like to play, check high scores, or look at credits. After selecting play, one must control a car to avoid traffic and other obstacles. A turret will be attached to the hood of the car which allows the player to launch a projectile and destroy the obstacles. Points are gained as time passes and as enemies are destroyed.

Noun extraction

Road rampage is a game in which firstly one is presented with a menu to select whether one would like to play, check high scores, or look at credits. After selecting play, one must control a car to avoid collision with traffic and other obstacles like shooters. A turret will be attached to the hood of the car which allows the player to launch a projectile and destroy the obstacles. Points are gained as time passes and as enemies are destroyed.

Entity classes:

Car, Collision, Traffic, Turret, Projectile, Shooters

The following diagram depicts how classes interact (the Shooters and Traffic Classes are depicted by the Obstacles Class)

The Keyboard Class Controls both the Car Class and Turret Class. And the Turret and
Car Class Communicates with the Obstacles and Projectile Class

**Boundary Class Extraction**

The boundary class which corresponds to the videogame will be a set of modules which will handle interaction and communication between the user and the software. Modules which correspond to the boundary class include the keyboard class which will detect when a keystroke is made, and will then make the corresponding changes to the game. Another module belonging to the boundary class will handle outputs to the screen and the graphics which will be loaded.

**Control Class Modeling**

The control class in the videogame is the collision controller because it is where entities will meet and this controller class will handle any interaction between the entities. What the collision controller class does is it detects any collisions made between the car class and the traffic, shooter or projectile classes and collisions made between the projectile class and traffic or shooter classes. The classes which must be checked for collision will send their coordinates to the collision controller, then the collision controller will decide whether a collision is made or not. The following diagram shows the updated class interactions with the Collision Class added.
10.3 Revised Dynamic Modeling

Every Class in the game, whether it is controller, boundary or entity, is constantly “waiting” for any changes to be made. Changes usually depend on interaction with other classes by constantly passing information to each other. When the information which the class was “waiting” for is passed, then a change is mad. The following diagram is a state chart which depicts how the passing of different information can change the state of a class and the game altogether.

State Chart

Through the main game even loop there will constantly be checks for any keystroke or for a collision. When either of these events occurs, the state changes. Depending on either if the collision occurred with the user’s car or between other objects then states of classes will change. Also depending on which button is pressed on the keyboard will cause states of other classes to be changed as well.

Communication Model

Throughout game play all classes are communicating with one another, sending information back and forth. Some information which is to be sent by a class depends on what information was first received by another class. It is very important to know how classes communicate because if any error was to occur it can easily be traced and targeted. The following diagram depicts a realization of the use-case “Move Car” which
was shown in the Requirements Workflow.
Section 11: Design Workflow 1st Iteration

11.1 Initial Component Design

CAR CLASS
Responsibilities:
- Declare variables to be used throughout the class
- Update position according to player’s inputs
- Move smoothly through the displaying stage
Collaboration:
- Rest of game
- Turret Class
- Collision Class

TURRET CLASS
Responsibilities:
- Declare variables to be used throughout the class
- Update position according to position of car or shooter
- Listen for input to shoot, and then shoot projectile accordingly
Collaboration:
- Rest of game
- Car Class
- Projectile Class
- Shooter Class

PROJECTILE CLASS
Responsibilities:
- Declare variables to be used throughout the class
- Travel smoothly through the displaying stage
- Update positions according to turret position and motion
Collaboration:
- Rest of game
- Turret Class
- Collision Class

COLLISION CLASS
Responsibilities:
- Get positions of vehicles throughout the game stage
- Display corresponding sprite when collision is detected
- Listen for collision
Collaboration:
- Rest of game
- Car class
- Shooter Class
- Traffic Class
- Projectile class
SHOOTER CLASS
Responsibilities:
  Update position according to function controlling its position
Collaboration:
  Rest of game
  Collision Class
  Turret Class

TRAFFIC CLASS
Responsibilities:
  Update Position according to the function controlling its position
Collaboration:
  Rest of game
  Collision Class
  Turret Class

CAR CLASS
- float: x, y,
- CBaseSprite: carsprite

Functions:
- Getxy()
- Setxy()
- CreateCar()
- LoadCar()
- DisplayCar()

TURRET CLASS
- float: x,y
- int: angle

Functions:
- Getxy()
- Setxy()
- SetAngle()
- CreateTurret()
- LoadTurret()
- DisplayTurret()
Projectile Class

Float: x, y, xspeed, yspeed
CBaseSprite: projectilesprite

Functions:
Getxy()
Setxy()
SetDestination()
SetInitiation()
CreateProjectile()
LoadProjectile()
DisplayTurret()

Collision Class

Float: x1, y1, x2, y2
CBaseSprite: collisionsprite

Functions:
CollisionCheck()
11.2 Initial Interface Design

There will be several methods which will build the interface of the game; these methods will be included in a class named the Game Class.

GAME CLASS:

<table>
<thead>
<tr>
<th>Method</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateObjects()</td>
<td>In this method, the modules which create the objects For every class will be called.</td>
</tr>
<tr>
<td>ProcessFrame()</td>
<td>This method is very important because it will check for any changes in the game for every frame. The CollisionCheck() module will be in this method.</td>
</tr>
<tr>
<td>LoadImages()</td>
<td>In this method, the modules which load the images for every class will be called.</td>
</tr>
<tr>
<td>Keyboard()</td>
<td>In this method the modules which listen for keystrokes will be called.</td>
</tr>
<tr>
<td>ComposeFrame()</td>
<td>In this method length of frames, fps and any audio to be added will be determined.</td>
</tr>
<tr>
<td>Display()</td>
<td>In this method all of the modules which display corresponding classes will be called. ComposeFrame() will also be called.</td>
</tr>
</tbody>
</table>
11.3 Initial Architectural Design
11.4 Initial Detailed Design

Various algorithms will be running through various data inputs at various points in time during the game. Here are the main algorithms which will be used in the game.

Description: Move car according to the player’s inputs.

Pseudo code: Begin
Input direction of movement
Get current coordinates of car
Check for restrictions
If no restrictions are found
Move car
Else
Don’t move car
End
Description: Check for collision between two objects

Pseudo code: Begin
    Get current position for both objects
    Calculate distance
    If distance is less than 20 pixels
        Collision occurs
    Else collision does not occur
End
Section 12: Review of Design Workflow by other Team Member

12.1 Review from Simranjeet Singh

Overview

Interface design I think should be in format of Actions, objects, and user, in a table. Still it explains as it is supposed to. Architectural design will improve if more detailed diagram of interactions of main interface is given. Detailed design should include: Method Name, Return Type, Input argument(s), Output argument(s), Methods invoked, Narrative for all classes along with pseudo code.

Preparation

A Lot is needed to be added. The things that are there are all right. In some cases diagrams can be more clear, redraw arrows to make it neat.

Inspection

Everything is to be fixed.
Section 13: Design workflow 2\textsuperscript{nd} Iteration

13.1 Revised Component Design

**CAR CLASS**

Responsibilities:
- Declare variables to be used throughout the class
- Update position according to players inputs
- Move smoothly through the displaying stage

Collaboration:
- Rest of game
- Turret Class
- Collision Class

**TURRET CLASS**

Responsibilities:
- Declare variables to be used throughout the class
- Update position according to position of car or shooter
- Listen for input to shoot, and then shoot projectile accordingly

Collaboration:
- Rest of game
- Car Class
- Projectile Class
- Shooter Class

**PROJECTILE CLASS**

Responsibilities:
- Declare variables to be used throughout the class
- Travel smoothly through the displaying stage
- Update positions according to turret position and motion

Collaboration:
- Rest of game
- Turret Class
- Collision Class

**COLLISION CLASS**

Responsibilities:
- Get positions of vehicles throughout the game stage
- Display corresponding sprite when collision is detected
- Listen for collision

Collaboration:
- Rest of game
- Car class
- Shooter Class
- Traffic Class
- Projectile class
SHOOTER CLASS
Responsibilities:
Update position according to function controlling its position
Collaboration:
Rest of game
Collision Class
Turret Class

TRAFFIC CLASS
Responsibilities:
Update Position according to the function controlling its position
Collaboration:
Rest of game
Collision Class
Turret Class
**Projectile Class**

- Float: x, y, xspeed, yspeed
- CBaseSprite: projectilesprite

  **Functions:**
- Getxy()
- Setxy()
- SetDestination()
- SetInitiation()
- CreateProjectile()
- LoadProjectile()
- DisplayTurret()

**Collision Class**

- Float: x1, y1, x2, y2
- CBaseSprite: collisionsprite

  **Functions:**
- CollisionCheck()
13.2 Revised Interface Design

There will be several methods which will build the interface of the game, these methods will be included in a class named the Game Class.

**GAME CLASS:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateObjects()</td>
<td>In this method, the modules which create the objects for every class will be called.</td>
</tr>
<tr>
<td>ProcessFrame()</td>
<td>This method is very important because it will check for any changes in the game for every frame. The CollisionCheck() module will be in this method.</td>
</tr>
<tr>
<td>LoadImages()</td>
<td>In this method, the modules which load the images for every class will be called.</td>
</tr>
<tr>
<td>Keyboard()</td>
<td>In this method the modules which listen for keystrokes will be called.</td>
</tr>
<tr>
<td>ComposeFrame()</td>
<td>In this method length of frames, fps and any audio to be added will be determined.</td>
</tr>
<tr>
<td>Display()</td>
<td>In this method all of the modules which display corresponding classes will be called. ComposeFrame() will also be called.</td>
</tr>
</tbody>
</table>
13.3 Revised Architectural Design
13.4 Revised Detailed Design

Various algorithms will be running through various data inputs at various points in time during the game. Here are the main algorithms which will be used in the game.

Description: Move car according to the players inputs.

Pseudo code: Begin
Input direction of movement
Get currents coordinates of car
Check for restrictions
If no restrictions are found
   Move car
Else
   Don’t move car
End

Description: Check for collision between two objects

Pseudo code: Begin
Get current position for both objects
Calculate distance
If distance is less than 20 pixels
   Collision occurs
Else collision does not occur
End

PDL for Collision:

```c
void collision(Car car, Traffic traffic, CBaseSprite * carsprite, CBaseSprite * trafficsprite, CBaseSprite * turretsprite, bool * gameover)
{
    float cx, cy, tx, ty, xb=30, yb=50;
    car.Getxy(cx, cy);
    traffic1.Getxy(tx, ty);

    if(((tx>=cx-yb) && (tx<=cx+yb)) && ((ty>=cy-xb) && (ty<=cy+xb)))
    {
        int r=255, g=255, b=255, frameCount=1, frame=0;
        carsprite->loadFrame(frame, "expl.bmp", r, g, b);
        carsprite->loadFrame(frame+1, "expl2.bmp", r, g, b);
    }
}
```
carsprite->loadGLTextures();
background.setDelRoll(0);

turretsprite->loadFrame(frame, "blank.bmp", r, g, b);
turretsprite->loadGLTextures();

trafficsprite->loadFrame(frame, "blank.bmp", r, g, b);
trafficsprite->loadGLTextures();

gameover = true;
Section 14: Review of Own Workflow

14.1 Self Review

Overview:
- CRC cards need to be properly designed and shown
- Operations required for communicating with classes needs to be properly shown
- Level of coupling needs to be shown
- Architectural design showing location of components needs to be shown
- All PDL’s and Flow charts need to be shown

Preparation:
- All CRC cards will be added and properly displayed
- Interface design will be properly depicted
- Architectural Design will be properly Depicted
- All components of the detailed design will be depicted
Section 15: Design workflow 3rd Iteration

15.1 Revised Component Design

The following diagrams are called Class Responsibility Collaboration Cards. CRC cards depict the responsibilities of the class along with a list of classes which it invokes to achieve those responsibilities.

### Car Class

**Responsibility:**
1. Send current coordinates to Keyboard Class
2. Receive updated coordinates from Keyboard Class
3. Send coordinates to Turret Class
4. Send coordinates to Collision Class

**Collaboration:**
1. Turret Class
2. Keyboard Class
3. Collision Class

### Turret Class

**Responsibility:**
1. Send Turret State to Projectile Class
2. Send Coordinates to Projectile Class
3. Receive Coordinates from Car Class
4. Receive Updated state from Keyboard Class
5. Update State

**Collaboration:**
1. Car Class
2. Keyboard Class
3. Projectile Class
### Projectile Class

**Responsibility:**
1. Send Coordinates to Collision Class
2. Receive Coordinates from Turret Class
3. Receive turret state from Turret Class
4. Receive Inputs from Keyboard
5. Move
6. Spawn

**Collaboration:**
1. Collision Class
2. Turret Class
3. Keyboard Class

### Shooter Class

**Responsibility:**
1. Send Coordinates to Collision Class
2. Spawn Initially
3. Re-spawn Randomly
4. Send Coordinates to Turret Class

**Collaboration:**
1. Collision Class
2. Turret Class

### Traffic Class

**Responsibility:**
1. Send Coordinates to Collision Class
2. Spawn Initially
3. Re-spawn Randomly

**Collaboration:**
1. Collision Class
## Collision Class

**Responsibility:**
1. Receive coordinates from Car Class
2. Receive coordinates from Projectile Class
3. Receive coordinates from Obstacles Class
4. Determine whether collision occurs
5. Send information to Graphics Modules

**Collaboration:**
1. Car Class
2. Projectile Class
3. Obstacles Class
4. Graphics Modules
15.2 Revised Interface Design

There will be several methods which will build the interface of the game, these methods will be included in a class named the Game Class.

GAME CLASS:

<table>
<thead>
<tr>
<th>Method</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateObjects()</td>
<td>In this method, the modules which create the objects for every class will be called.</td>
</tr>
<tr>
<td>ProcessFrame()</td>
<td>This method is very important because it will check for any changes in the game for every frame. The CollisionCheck() module will be in this method.</td>
</tr>
<tr>
<td>LoadImages()</td>
<td>In this method, the modules which load the images for every class will be called.</td>
</tr>
<tr>
<td>Keyboard()</td>
<td>In this method the modules which listen for keystrokes will be called.</td>
</tr>
<tr>
<td>ComposeFrame()</td>
<td>In this method length of frames, fps and any audio to be added will be determined.</td>
</tr>
<tr>
<td>Display()</td>
<td>In this method all of the modules which display corresponding classes will be called. ComposeFrame() will also be called.</td>
</tr>
</tbody>
</table>
The following diagram shows the final detailed class diagram.

Levels of Coupling:

Level of coupling is always attempted to keep low in order to promote reusability of the modules. In the way in which the classes are presented is the optimum positions for the lowest coupling.
15.3 Revised Architectural Design

The diagram tells the levels of cohesion for every module in the software product.
15.4 Revised Detailed Design

Various algorithms will be running through various data inputs at various points in time during the game. Here are the main algorithms which will be used in the game.

Description: Move car according to the players inputs.

Pseudo code: Begin
   Input direction of movement
   Get currents coordinates of car
   Check for restrictions
   If no restrictions are found
      Move car
   Else
      Don’t move car
End

PDL for Move Car:

void keyOperations (void)
{
   float x, y;
   car.Getxy(x,y);
if(!gameover && !pause.on)
{
    if(!keyStates['s'])
    {
        y += 1;
        if (y > 410)
            y = 410;
    }
    if(!keyStates['w'])
    {
        y -= 1;
        if (y < 100)
            y = 100;
    }
    if(!keyStates['a'])
    {
        x -= 1;
        if (x < 0)
            x = 0;
    }
    if(!keyStates['d'])
    {
        x += 1;
        if (x > background.getWidth())
            x = background.getWidth();
    }
}

Description: Check for collision between two objects

Pseudo code:  Begin
    Get current position for both objects
    Calculate distance
    If distance is less than 20 pixels
        Collision occurs
    Else collision does not occur
PDL for Collision:

```c
void collision(Car car, Traffic traffic, CBaseSprite * carsprite, CBaseSprite * trafficsprite, CBaseSprite * turretsprite, bool * gameover)
{
    float cx, cy, tx, ty, xb=30, yb=50;
    car.Getxy(cx, cy);
    traffic1.Getxy(tx, ty);

    if(((tx>=cx-yb) && (tx<=cx+yb)) && ((ty>=cy-xb) && (ty<=cy+xb)))
    {
        int r=255, g=255, b=255, frameCount=1, frame=0;
        carsprite->loadFrame(frame, "expl.bmp", r, g, b);
        carsprite->loadFrame(frame+1, "expl2.bmp", r, g, b);
    }
}```
carsprite->loadGLTextures();
background.setDelRoll(0);

turretsprite->loadFrame(frame, "blank.bmp", r, g, b);
turretsprite->loadGLTextures();

trafficsprite->loadFrame(frame, "blank.bmp", r, g, b);
trafficsprite->loadGLTextures();

gameover = true;

Description: Shoot a projectile at the players command

Pseudo code:

Begin

Receive the players input
Receive the turret’s coordinates
Spawn and Move
Send Coordinated to collision class
If collision
    Update Graphics
Else
    Remove sprite

End
PDL for Shoot:

void keyOperations(void)
{
    if(!keyStates[ ' '])
    {
        float xb, yb, xt, yt;

        if(up)
        {
            turret.Getxy(xt,yt);
            xb = xt - 24;
            yb = yt - 16;
            bulletUP[b1].Setxy(xb,yb);
            b1++;
            if(b1 == bulletNum-1)
            {
                b1=1;
            }
        }
    }
}
keyStates[ ' ' ] = true;
}
} else if(down)
{
turret.Getxy(xt, yt);
xb = xt - 27;
yb = yt + 13;
bulletDOWN[b3].Setxy(xb, yb);
b3++;
if(b3 == bulletNum - 1)
{
    b3 = 1;
}
keyStates[ ' ' ] = true;
} else
{
turret.Getxy(xt, yt);
xb = xt - 36;
yb = yt - 2;
bullet[b2].Setxy(xb, yb);
b2++;
if(b2 == bulletNum - 1)
{
    b2 = 1;
}
keyStates[ ' ' ] = true;
}
Description: Have the shooters obstacles loop in the game

Pseudo code:
Begin
    Initiate the shooter by traveling along a y = int line
    Send coordinates to collision class
    Receive update from collision class
    If Collision
        Update Graphics
        Randomize new position
    Else
        Keep traveling
End
Description: Turn the turret at the players command

Pseudo code:
Begin
    Receive Inputs from keyboard
    Check for boundary restriction
    If restriction
        Do not turn turret
    Else
        Turn Turret
End

PDL for Turn Turret

void keyOperations(void)
{
    if(!keyStates['o'])
    {
        if(down)
        {
            int r=255, g=255, b=255, frameCount=1, frame=0;
            turretsprite->loadFrame(frame, "turret2.bmp", r, g, b);
            turretsprite->loadGLTextures();
            down = false;
            keyStates['o'] = true;
        }
        else
        {
            // Code
        }
    }
}
int r=255, g=255, b=255, frameCount=1, frame=0;
turretsprite->loadFrame(frame, "turret3.bmp", r, g, b);
turretsprite->loadGLTextures();
up = true;
keyStates['o'] = true;
}
}
if(!keyStates['k'])
{
    if(up)
    {
        int r=255, g=255, b=255, frameCount=1, frame=0;
turretsprite->loadFrame(frame, "turret2.bmp", r, g, b);
turretsprite->loadGLTextures();
up=false;
keyStates['k'] = true;
    }
    else
    {
        int r=255, g=255, b=255, frameCount=1, frame=0;
turretsprite->loadFrame(frame, "turret5.bmp", r, g, b);
turretsprite->loadGLTextures();
down = true;
keyStates['k'] = true;
    }
}
}
Section 16: Execution Based Test design 1\textsuperscript{st} Iteration

16.1 Initial Black Box Testing

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Press a</td>
<td>Car moves forward</td>
</tr>
<tr>
<td>Press o</td>
<td>Turret Rotates up</td>
</tr>
<tr>
<td>Press ‘Spacebar’</td>
<td>A projectile is shot</td>
</tr>
</tbody>
</table>

16.2 Initial White Box Testing

1. 1 2 4
2. 1 2 5 7 8
3. 1 3 5 7 8
4. 1 3 6

Path one will be tested by having an enemy shoot, then miss and the shoot sprite should be removed. Path two will be tested by having an enemy shoot, then hit the player and loosing health then ending the game when health is empty. Path 3 will be tested by having an enemy car collide with the player’s car and loose health until health is empty, which should end the game. And path 4 will be tested by missing the player with an enemy car then removing the sprite from the screen.
Section 17: Review of Execution Based Test Design by other team member

17.1 Review by Simranjeet Singh
    Overview:
    All possible paths seem to be included. Paths are well explained.
    Preparation:
    1. Very little black box testing is shown, needs more detail.

    Inspection:
    Complete black-box testing will be done
Section 18: Execution Based Test Design 2\textsuperscript{nd} Iteration

18.1 Revised Black Box Testing

Input: Expected output:

- Press a: Car moves forward
- Press o: Turret Rotates up
- Press ‘Spacebar’: a projectile is shot
- Press a and w: Car moves diagonally
- Press k: Turret Rotates Down
- Press p: The Game is Paused
- Press 0: The Game Exits
- Press a, w, o and Spacebar: The car moves diagonally, the turret is rotated upward, and a projectile is shot

18.2 Revised White Box Testing

1. 1 2 4
2. 1 2 5 7 8
3. 1 3 5 7 8
4. 1 3 6

Path one will be tested by having an enemy shoot, then miss and the shoot sprite should be removed. Path two will be tested by having an enemy shoot, then hit the player.
and looseing health then ending the game when health is empty. Path 3 will be tested by having an enemy car collide with the player’s car and loose health until health is empty, which should end the game. And path 4 will be tested by missing the player with an enemy car then removing the sprite from the screen.
Section 19: Review Own Execution Based Test Design

19.1 Self Review
   - Black box testing still needs work, more testing can be added
   - White box testing looks good
Section 20: Execution Based Test Design 3rd Iteration

20.1 Revised Black Box Testing

Input:  
Press a  
Press o  
Press ‘Spacebar’  
Press a and w  
Press k  
Press p  
Press 0  
Press a, w, o and Spacebar  
Collide with enemy

Expected output:  
Car moves forward  
Turret Rotates up  
A projectile is shot  
Car moves Diagonally  
Turret Rotates Down  
The Game is Paused  
The Game Exits  
The car moves diagonally, the turret is rotated upward, and a projectile is shot  
Both objects are destroyed and the game ends

20.2 Revised White Box Testing

```
1. 1 2 4  
2. 1 2 5 7 8  
3. 1 3 5 7 8  
4. 1 3 6
```
Path one will be tested by having an enemy shoot, then miss and the shoot sprite should be removed. Path two will be tested by having an enemy shoot, then hit the player and loosing health then ending the game when health is empty. Path 3 will be tested by having an enemy car collide with the player’s car and loose health until health is empty, which should end the game. And path 4 will be tested by missing the player with an enemy car then removing the sprite from the screen.
Section 21: Results of Execution Based Tests

21.1 Black Box Test Results
   All expected outputs were satisfied by the inputs

21.2 White Box Test Results
   White box testing was not done due to the absence of the Drivers
Section 22: Conclusion

The complete software engineering process has been understood completely and can now be put to practice on other software to be developed in the future. It is now understood why it is necessary for various iterations to take place for all workflows. In order for a software product to be made correctly, it is important to optimize its design and do rigorous reviewing before implementation because the product must work properly in every aspect before it is released.