**INTRODUCTION**

* 1. **Background and Objectives**

As per the ENGG 102 course requirement of the first year second semester of Computer Engineering, we created a car game called ‘RUSH’ using C programming language. The game depends upon SDL package for its graphics. Our game is modified version of the flash game called ‘AUTOBAHN’.

The objective of our game is to run away from the cop chasing us. We have implemented similar concept for our game with some major and minor modification. Modifications are discussed latter in the section.

‘RUSH’ is a single player 2D car racing game consisting features like cop chase, fuel indicator, scores and traffics as obstacles. Besides these features, we have included graphics and sound effects to our game.

**DISCUSSION**

* 1. **The C Language**

Different programming languages are used for programming games. Out of them C programming language is most often chosen by the programmers due to following features provided by it:

* It is easy use and understand
* It provides easy support for flexible and extensible style of programming.
* It has very limited restrictions when it comes to hardware’s access and utilizations. This enable user to directly manipulate devices for interactive gaming.
* Pointers are very powerful tools of C language that enable game programmers to control various events and actions during the game.
* Easy to use file handling system for good score management system.

But it has some limitations also:

* Lack of real world modeling of programming problem makes it difficult to implement big programs easily.
* Globalization of variables and unrestricted access to them makes the program hard to debug and test.
* Although pointer are strong hold of C, but incorrect use to them may lead to disastrous result.

By far it outweighs all other language when it comes to developing 3D games. Almost all high-end games are programmed in C today. Thus considering all these features we decided to use C programming language for our project.

* 1. **Resources Used**

Besides using C programming language for our game development we used other resource for various purposes in the game. They are described in details below:

* + 1. *Simple Direct Media Layer(SDL)*

Standard library that is provided by the C compiler manufacturer doesn’t provide enough support for good game development. The graphics and sound support are poor. And handling of user events becomes a difficult task.

SDL is a free open source game development library that is entirely coded in C programming language. It is written over Open Graphics Library (OpenGL), this help to simplify OpenGL providing easy functionality to handle its function and structures. SDL is a composite of many small libraries that can be incorporated together to develop games. Various SDL libraries implements in our game are listed below with their implementation in the game.

|  |  |  |
| --- | --- | --- |
| S.N. | SDL Library Component | Implementation in the game |
| 1. | SDL | It is the main library of the SDL package as is used for generating main screen, event handling, for SDL structures like SDL\_Rect, SDL\_Color, SDL\_Surface. Other implementation of this library is for images handling. |
| 3. | SDL\_Mixer | This Library provides support for sounds and music effect for the game. It has function like *Mix\_PlayMusic(), Mix\_HaltMusic()* and other functions that enables us to control sound play in the game. Besides it also provides with the *Mix\_Music* and *Mix\_Chunk* structures that are used for holding game music and sound effect. |
| 4. | SDL\_Font | This library provides us with the support for fonts and displaying messages in the game. It has TTF\_Font structure for loading fonts and functions like TTF\_RenderText\_Blender() for displaying the texts. |

Table : Componets of SDL Library used in the game

* + 1. *Sound Editing*

Editing of sound and music clips was necessary to cut out noises and non-essential part of the music that intervened during the game play. In the mean time it was also necessary to convert all music and sound format to standard WAV and MP3 formats. These music formats are supported by the SDL Library. The purpose of sound editing and conversion was accomplished using *Total Converter*.

2.3 **Project Implementation**

* + 1. *Adding and manipulating Graphics*

Graphics is the essential part of our game. We custom designed our graphics by using graphics editing software’s. The main challenge was to incorporate these graphics into our game. This was accomplished by using following SDL Library functions:

|  |
| --- |
| **Function Name:** SDL\_LoadBMP  **Number of Arguments:** 1  **Return Type:** SDL\_Surface \*  **Return Value:** Return a pointer to the image after loading the file into computer memory.  **Arguments Description:**  Argument 1: Locations of the image file to be loaded.  **Function’s Description:**  This is a SDL image library function that loads image from the specified file into computer memory. On successful loading of file it return a pointer pointing to the location in the memory where the image has been loaded. This pointer is then assigned to the image pointer variable in the game and then the image is manipulated using this pointer. If it fails to load the image it returns a NULL pointer.  **Sample Implementation:**  SDL\_LoadBMP("images/car.bmp"); |
| **Function Name:** SDL\_DisplayFormat()  **Function prototype:** SDL\_Surface \* SDL\_DisplayFromat(SDL\_Surface \*image);  **Number of Arguments:** 1  **Return Type:** SDL\_Surface \*  **Return Value:** Return a pointer to the image after optimization.  **Arguments Description:**  Argument 1: The pointer to the image that is to be optimized  **Function’s Description:**  This is a SDL image library function that optimizes a given image (already loaded into computer memory) to the format of the game display.  **Sample Implementation:**  SDL\_Surface \*analyze\_image(SDL\_Surface \*src)  {  //Temporary sdl surfaces  SDL\_Surface \*loaded\_image; //to hold the loaded image  SDL\_Surface \*optimized\_image; //to hold the optimized image    loaded\_image=src;    //optimize the image  optimized\_image= SDL\_DisplayFormat(loaded\_image);    //Free the temporary buffer  SDL\_FreeSurface(loaded\_image);    //return the optimized image  return optimized\_image;    } |
| **Function Name:** SDL\_SetColorKey  **Function prototype:** int SDL\_SetColorKey(SDL\_Surface \* image, SDL\_SRCCOLORKEY, SDL\_Color RGB);  **Number of Arguments:** 3  **Return Type:** int  **Return Value:** Return 0 if the conversion was successful else return -1 if there was an error.  **Arguments Description:**  Argument 1: Image pointer that is to be color keyed.  Argument 2: SDL flag that tells the function to make the pixels of given color transparent.  Argument 3: SDL color structure defining the color of the pixels that is to be made transparent specified with Red, Green and Blue values.  **Function’s Description:**  This is a SDL image library function that makes pixels of given color transparent and then return this modified image as a pointer. |
| **Function Name:** SDL\_FreeSurface  **Function prototype:** void SDL\_FreeSurface(SDL\_Surface \*image);  **Number of Arguments:** 1  **Return Type:** void  **Arguments Description:**  Argument 1: The image pointer pointing to the image that is to be freed from the computer memory  **Function’s Description:**  This is a SDL image library function that frees the resources used by a given image in the computer memory. This function is called at the end of the game when the images that have been previously loaded in the game are to be freed in order to free the resources used by the game and prevent Memory Leak.  **Sample Implementation:**  void free()  {  SDL\_FreeSurface(background);  SDL\_FreeSurface(image);  } |

Here is a snapshot from the game screen showing various images that has been loaded in the game.

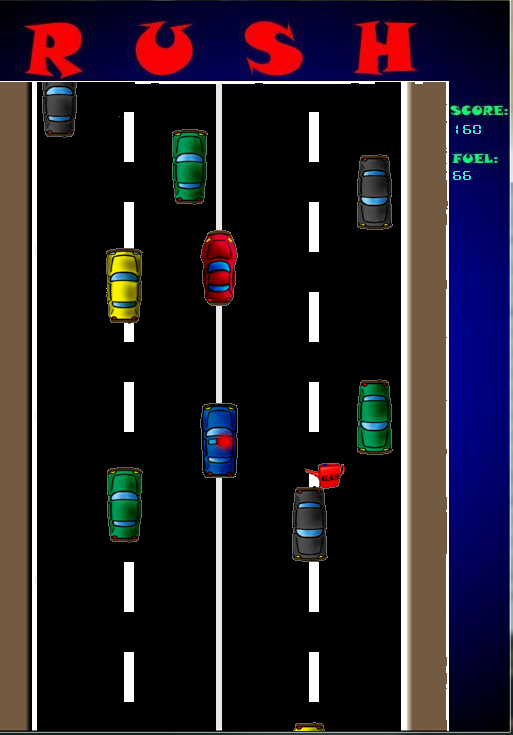


Figure : Graphics loaded in the game

* + 1. *Adding Sounds*

Sound effects were required for various events in the game. The Main background music plays with the start of the game. Other small sound effect like collision and police siren effect all were implemented in the game by using the SDL Mixer library. This library provides support for loading, playing and manipulating sounds as per requirement. Adding of sound by using the SDL mixer library was done using following functions:

|  |
| --- |
| **Function Name:** Mix\_LoadMUS  **Function prototype:** Mix\_Music \* Mix\_LoadMUS( const char \* file\_name);  **Number of Arguments:** 1  **Return Type:** Mix\_LoadMUS  **Return Value:** Return a pointer to the music after loading the file into computer memory.  **Arguments Description:**  Argument 1: Locations of the music file to be loaded.  **Function’s Description:**  This is a SDL mixer library function that loads music from the specified file into computer memory. On successful loading of file it return a pointer pointing to the location in the memory where the music has been loaded. This pointer is then assigned to the music pointer variable in the game and then the music is manipulated using this pointer. If it fails to load the music it returns a NULL pointer.  **Sample Implementation:**  background\_music = Mix\_LoadMUS("sounds\\music.wav");  click\_sound = Mix\_LoadWAV("sounds\\siren.wav");  wall\_explode = Mix\_LoadWAV("sounds\\collide.wav"); |
| **Function Name:** Mix\_PlayMusic()  **Function prototype:** int Mix\_PlayMusic(Mix\_Music \*music, int repetitions);  **Number of Arguments:** 2  **Return Type:** int  **Return Value:** Returns 0 on success else returns -1 indicating error in music play.  **Arguments Description:**  Argument 1: Pointer to the music file that is to be played  Argument 2: Number of times to play the music, -1 means to repeat the music for indefinite period.  **Function’s Description:**  This is a SDL mixer library function that allows us to play the given music file for desired number of times.  **Sample Implementation:**  Mix\_PlayMusic(background\_music,-1); |
| **Function Name:** Mix\_LoadWAV  **Function prototype:** Mix\_Chunck \* Mix\_LoadWAV(char \*file\_name);  **Number of Arguments:** 1  **Return Type:** Mix\_Chunk\*  **Return Value:** Return the pointer location to the WAV file loaded in the memory.  **Arguments Description:**  Argument 1: Location of the WAV music file.  **Function’s Description:**  This is a SDL Mixer library function that allows us to load small music effect require for the game. |
| **Function Name:** Mix\_FreeMusic  **Function prototype:** void Mix\_FreeMusic(Mix\_Music \*music);  **Number of Arguments:** 1  **Return Type:** void  **Return Value:** Nothing  **Arguments Description:**  Argument 1: The music pointer pointing to the location where the music file is loaded.  **Function’s Description:**  This is a SDL image library function that frees the resources used by a given music in the computer memory. This function is called at the end of the game when the music that have been previously loaded in the game are to be freed in order to free the resources used by the game and prevent Memory Leak.  **Sample Implementation:**  Mix\_FreeMusic(bg\_music);  Mix\_FreeChunk(siren);  Mix\_FreeChunk(colide); |

* + 1. *Objects motions and control*

Player, cop and traffics are moved using various functions. Player is moved using keyboard. Cop is moved using AI and traffics are generated and moved by functions based on their current position.

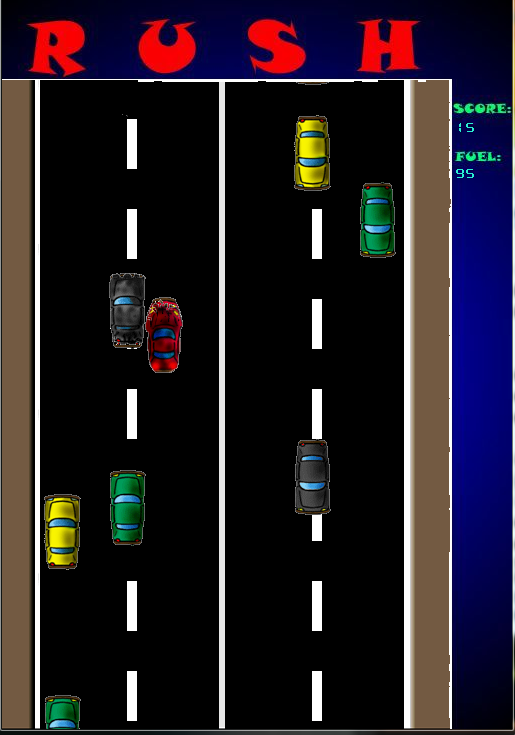
All these motions and controls are implemented in the game using following functions:

|  |
| --- |
| **Function Name:** move\_car  **Function prototype:** void move\_car(SDL\_Rect &a,int direction)  **Number of Arguments:** 2  **Return Type:** void  **Arguments Description:**  Argument 1: A pointer to the player structure variable that is to be moved  Argument 2: Numerical value indicating the direction in which the player is to be moved.  **Function’s Description:**  This function moves the given player in given direction. The function checks the second argument of the function which it treats as direction and then the object denoted by the second argument is moved in the given direction.  Here directions up, down, left, right are ddefined as 1,2,3,4 respectively.  **Actual Implementation:**  void move\_car(SDL\_Rect &a,int direction)  {  if(direction==UP)  {  a.y-=3;  }  else if(direction==DOWN)  {  a.y+=3;  }  else if(direction==LEFT)  {  a.x-=2;  }  else if(direction==RIGHT)  {  a.x+=2;  }  } |
| **Function Name:** move\_cop  **Function prototype:** void move\_cop(SDL\_Rect &a,SDL\_Rect b,SDL\_Rect c) **Number of Arguments:** 3  **Return Type:** void  **Arguments Description:**  Argument 1: Pointer to the bullet structure variable that is to be moved.  Argument 2: Pointer to the bullet structure variable that is to be compared.  Argument 3: Pointer to the bullet structure variable that is to be followed.  **Function’s Description:**  This function gives the condition for the movement of the cop. This function moves the cop in forward direction following the car after comparing its position with the traffics. Function first compares the cops position with the traffic and then with car and does the necessary movements.  **Implementation:**  void move\_cop(SDL\_Rect &a,SDL\_Rect b,SDL\_Rect c)  {  if((a.x==(b.x+41)) && (a.y>=(b.y-82) && a.y<=(b.y+82)))  {  if(a.x<c.x)  {  a.x+=1;  }  }  else if((a.x==(b.x-41)) && (a.y<(b.y+82) && a.y>(b.y-82)))  {  if(a.x>c.x)  {  a.x-=1;  }  }  else  {  if(a.x<c.x)  {  a.x+=1;  }  else if(a.x>c.x)  {  a.x-=1;  }  else if(a.x==c.x)  {  a.x=c.x;  }  }  } |
| **Function Name:** position\_traffic  **Function prototype:** void position\_traffic\_down(TRAFF &T, int j);  **Number of Arguments:** 2  **Return Type:** void  **Arguments Description:**  Argument 1: The pointer to the enemy structure variable that is to be moved.  Argument 2: The id of the object  **Function’s Description:**  This function gives the initial position of the traffic cars along with its speed. It uses switch statements with id as the case and gives the positions accordingly.  **Actual Implementation:**  void position\_traffic\_down(TRAFF &T, int j)  {  switch (j)  {  case 0:  {  T.speed=5;  T.TRAFFIC.y=50;  T.TRAFFIC.x=225;  break;  }  case 1:  {  T.speed=4;  T.TRAFFIC.y=391;  T.TRAFFIC.x=290;  break;  }  case 2:  {  T.speed=3;  T.TRAFFIC.y=300;  T.TRAFFIC.x=355;  break;  }  case 3:  {  T.speed=5;  T.TRAFFIC.y=400;  T.TRAFFIC.x=225;  break;  }  }  } |
| **Function Name:** reposition\_traffic  **Function prototype:** void reposition\_traffic\_down(TRAFF &T, int j)  **Number of Arguments:** 2  **Return Type:** void  **Arguments Description:**  Argument 1: Pointer to the traffic structure  Argument 2: id of the traffic  **Function’s Description:**  The function gives the traffic new position and speed once it goes out of the game screen.  **Implementation:**  void reposition\_traffic\_down(TRAFF &T, int j)  {  switch (j)  {  case 0:  {  if(T.TRAFFIC.y>750)  {  T.TRAFFIC.y=0;  switch (T.TRAFFIC.x)  {  case 225:  {  T.TRAFFIC.x=290;  T.speed=5;  break;  }  case 355:  {  T.TRAFFIC.x=225;  T.speed=4;  break;  }  case 290:  {  T.TRAFFIC.x=355;  T.speed=3;  break;  }  }  }  .  .  .  } |
| **Function Name:** check4cop  **Function prototype:** int check4cop(SDL\_Rect C)  **Number of Arguments:** 1  **Return Type:** int  **Return Value:** Returns the id of the traffic near to cop  **Arguments Description:**  Argument 1: Pointer to the cop structure  **Function’s Description:**  This function simply checks which of the traffic is near to the cop and returns its id.  **Implementation:**  int check4cop(SDL\_Rect C)  {  for(int i=0;i<4;i++)  {  if(T\_up[i].TRAFFIC.y>=(C.y-83) && T\_up[i].TRAFFIC.y<=(C.y+83))  {  return i;  }  }  } |

* + 1. *Collision detection*

Collision detection is the crucial part of the game and proved to be a big challenge for us initially. But however the problem was solved solved by implementing the following function:

|  |
| --- |
| **Function Name:** check\_collision  **Function prototype:** int check\_collision(SDL\_Rect a, SDL\_Rect a);  **Number of Arguments:** 2  **Return Type:** int  **Return Value:** Return 1(true) if the collision between the objects A and Object B is detected else return 0(false).  **Arguments Description:**  Argument 1: A SDL\_Rect structure variable holding the coordinates and dimension of the first object  Argument 2: A SDL\_Rect structure variable holding the coordinates and dimension of the second object  **Function’s Description:**  This two parameters of this function are taken as two rectangles and this function checks whether these two rectangles overlap each other or not and returns accordingly. It returns true(1) if the collision is detected and 0(false) if not detected.  **Algorithm:**  The necessary algorithm behind implementation of this function is:   1. Determine the offsets of each object top,down, left, right. 2. Compare to see if the left of first object is greater than right of the second object in that case return false. 3. Compare to see if the right of the first object is less than left of the second object in that case return false. 4. Compare to see if the top of first object is greater than bottom of second object in that case return false. 5. Compare to see if the bottom of first object is less than the top of second object in that case return false. 6. Finally is all above condition are not met than return true(collision detected).   **Actual Implementation:**  int check\_collision(SDL\_Rect a, SDL\_Rect b)  {  //The attributes of first object  int left\_a,right\_a;  int top\_a,bottom\_a;  //The attributes of first object  int top\_b,bottom\_b;  int left\_b,right\_b;  //Calculate the sides  left\_a=a.x;  right\_a=a.x+a.w;  top\_a=a.y;  bottom\_a=a.y+a.h;  //Calculate the sides  left\_b=b.x;  right\_b=b.x+b.w;  top\_b=b.y;  bottom\_b=b.y+b.h;  //test for conditions for the collision  if(bottom\_a>=top\_b)  {  if(right\_b<(right\_a+b.w) && right\_b>(right\_a-a.w))  {  if(bottom\_b>=top\_a)  {  return TRUE;  }  }  }  //If test fails  return FALSE;  } |



**Figure 2: Snapshot for collision**

* + 1. *Event Handling*

Event handling is the process of assessing events generated by the user during the game play. User generated events in the game includes key presses, button clicks, window events (close button click), joysticks motions etc. These events need to be handled in the game for enabling user interaction with the game.

Event handling has been done using the function and structure of the SDL library. SDL includes an Event structure. All the user events generated during the game play is arranged by SDL in a queue, first in first out. We then use SDL\_PollEvent() function to poll the next event in the queue and SDL assigns it to our event structure. Then by manipulating this event structure we handle the user generated events.

Detail implementation of the event handling is given below:

|  |
| --- |
| **Function Name:** SDL\_PollEvent  **Function prototype:** int SDL\_PollEvent(SDL\_Event \*event);  **Number of Arguments:** 1  **Return Type:** int  **Return Value:** Returns 1 if there is any pending event else return 0  **Arguments Description:**  Argument 1: A pointer to SDL\_Event structure variable which is assigned the event information by the function  **Function’s Description:**  This is a SDL Library function that is used to poll any pending user generated event. If any event is left than the specific information of the event is assigned to the event structure. This structure can then be check for what event has been generated and then manipulate the information accordingly. |

* + 1. *Score management*

In any game score are maintained to track the efficiency of the player playing the game. This also assist player to maintain a track of the gaming skills and to compare it with other player. This adds excitement and fun to the game.

Score in our game is recorded for two teams during the game play. Team scores when their player destroys walls, bricks, other opponent player or seal. This score is updated on the score board at the end of each level. At the end of the game, out of two scores top score is selected. This top score is then compared with the top 10 scores recorded in the “Scores.dat” file. If it falls within the top 10 scores then users are requested to enter their team name. This information is then saved to the same file for future reference.

Whole purpose of score management is accomplished using following function:

|  |
| --- |
| **Function Name:** Read Score  **Function prototype:** void read\_score();  **Number of Arguments:** 0  **Return Type:** void  **Return Value:** Nothing  **Arguments Description:**  (None)  **Function’s Description:**  This function reads top 10 scores from the file assigning them to the SCORES structure variables of the game.  **Algorithm:**  Algorithm for the read\_score() function is given below:   1. Open the file “Scores.dat” 2. Iterate a loop 10 times and read the scores from the file. 3. Assign the scores to variables in the game created to hold the scores 4. Close the file   **Actual Implementation:**  void read\_score()  {  //Our file pointer  FILE \*fptr;    //For looping  int i=0;    //Open the file Scores.dat  fptr=fopen("Scores","r");    //loop  for(i=0;i<10;i++)  {  //read the file and assign the score to the structure  fread(&score[i],sizeof(SCORES),1,fptr);    }  //close the file  fclose(fptr);  } |
| **Function Name:** Write Score  **Function prototype:** void write\_score(SCORES array[]);  **Number of Arguments:** 1  **Return Type:** void  **Return Value:** Nothing  **Arguments Description:**  Argument 1: A array of 10 SCORES structure variables containing all the top 10 scores.  **Function’s Description:**  This function writes the top 10 scores for the game in the file “Scores.dat”. Top 10 scores are stored in 10 SCORES structure variables in descending order.  **Algorithm:**  Algorithm for the read\_score() function is given below:   1. Open the file “Scores.dat” 2. Iterate a loop 10 times. 3. Write the score information to the file. 4. Close the file   **Actual Implementation:**  void write\_score(SCORES st[])  {  //Our file pointer  FILE \*fptr;    //for looping  int i=0;    //open the file  fptr=fopen("Scores","w");    //loop 10 times  while(i<=10)  {  //write the score information to the file  fwrite(&st[i],sizeof(SCORES),1,fptr);  //increment i  ++i;  }    //close the file  fclose(fptr);  } |
| **Function Name:** Sort Scores  **Function prototype:** void sort\_score();  **Number of Arguments:** 0  **Return Type:** void  **Return Value:** Nothing  **Arguments Description:**  (None)  **Function’s Description:**  This function sorts all the scores stored in the SCORES structure variable using a modified bubble sort. Finally after sorting the variables these variables are writing to the file “Scores.dat”. Sorting is done in the descending order.  **Algorithm:**  Algorithm for the read\_score() function is given below:   1. Iterate a loop 10 times 2. Sort the scores in the loop using bubble sort algorithm 3. Open the file “Scores.dat” 4. Iterate a loop 10 times. 5. Write the sorted score information to the file. 6. Close the file   **Actual Implementation:**  void sort\_score()  {  //Our file pointer  FILE \*fp;  //For looping  int i=0,j;  //Loop for 10 times using bubble sort algorithm  for( i=0;i<10;++i)  {  for(j=i;j<10;++j)  {  if(score[i].score < score[j].score)  {  //swapping  SCORES temp;  temp=score[i];  score[i]=score[j];  score[j]=temp;  }  }  }  //open the file to write sorted scores  fp=fopen("Scores","w");  //loop for writing score to the file  for(i=0;i<10;++i)  {  fwrite(&score[i],sizeof(SCORES),1,fp);  }  //close the file  fclose(fp);  } |

* 1. **User Interface Design**
     1. *Main Welcome Screen*

The main welcome screen welcomes the user to the game. It is of 510 pixels X 730 pixels. It has options such as to start or exit from the game. They are described in the table below with their function.

|  |  |
| --- | --- |
| Option | Function |
| START | Starts the game. (used by pressing ‘s’) |
| EXIT | Exits the game. (used by pressing ‘e’) |
| HIGHSCORES | Shows the scores. (used by pressing ‘h’) |

Table : Button in the main screen with their use

Here is a snapshot of the welcome screen:



Figure 7: Snapshot of the welcome/startup screen

* + 1. *Main Game Screen*

This is screen where the game begins. All the objects and elements of the game are provided in the screen. The positioning of the traffic and the movement of the car and cop is done using functions as described in the implementation section. It measures 510 pixels X 730 pixels. Here is a snapshot of the main game screen:

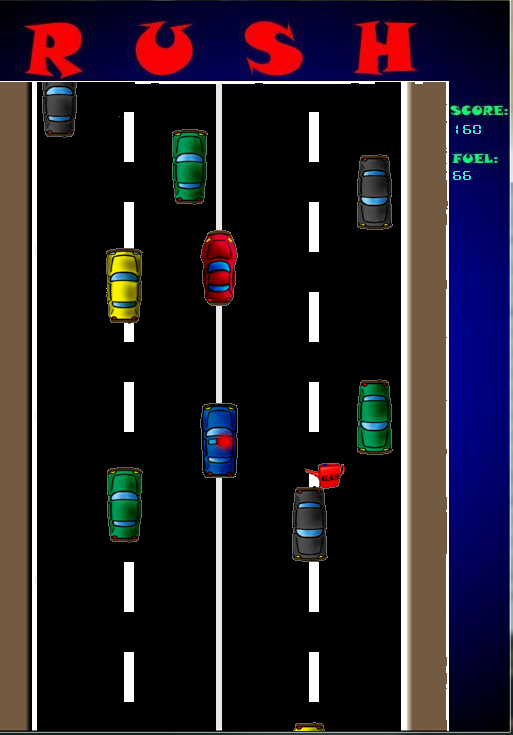


Figure 8: Main game screen with all the game objects

* + 1. *Score Board*

The Score board maintains the score and the fuel amount of the player and is located at the right hand side of the screen. The score gets automatically updated till the player is able to maintain its distance from the cop and until the fuel isn’t empty.

**GANTT CHART**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Work | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 | Week 10 | |
| Game Analysis and Design |  | | |  | | | | | | | |
| Graphics and sound programming |  |  | | | | | | | | |  |
| File handling |  | | | | |  |  | | | | |
| Event handling and player control |  |  | | | |  | | | | | |
| Packaging and testing |  | | | | | | | | |  | |

Table 6: Gantt char for the project

Week 1 started from – March 14, 2010

**CONCLUSION**

Our project to develop “RUSH” helped us learn a great deal about programming and team work. We are very grateful to all those who shared their ideas with us in developing this game, especially to our Supervisor Mr. Pankaj Raj Dawadi for letting us to carry out this project.

This project led us to some serious application development skills that we were totally unfamiliar with. We learnt the importance of planning and organization for building big software. As for a first year computer engineering student, we believe that we have put our best effort in completing this project.

We eagerly look forward for more such future endeavors challenging the limitations of our imaginations.

**BIBLOGRAPHY**

1. Programming with C, E.Balaguruswamy, Tata Mc.Graw hill publication.
2. SDL Library resource, [www.libsorg.com/SDL](http://www.libsorg.com/SDL)
3. SDL Tutorials [www.stdltutorial.com/index.php](http://www.stdltutorial.com/index.php)
4. Dev Paks Org, Resources to download Packages SDL Library for Dev Cpp, [www.devpaks.org/SDL](http://www.devpaks.org/SDL)
5. Lazy Foo, Begginers tutorials on SDL Library, [www.lazyfoo.net/SDL/tutorials](http://www.lazyfoo.net/SDL/tutorials)
6. <http://flashgamesite.com/play300game.html>
7. <http://www.dailyfreegames.com/games/racing-games/autobahn-game.html>

**APPENDICES**

**APPENDIX A:**

The Pictorial representation of the objects as used in the game.

|  |  |  |
| --- | --- | --- |
| Object Name | Pictorial Representation |  |
| Player | car.bmp car_damaged.bmp  cop.bmp cop_damaged.bmp  traffic0.bmp traffic0_damaged.bmp  gas.bmp | |
| Cop |
| Traffic |
| Wall |
| Fuel |

Table 7: Table of Game objects with symbolic representation