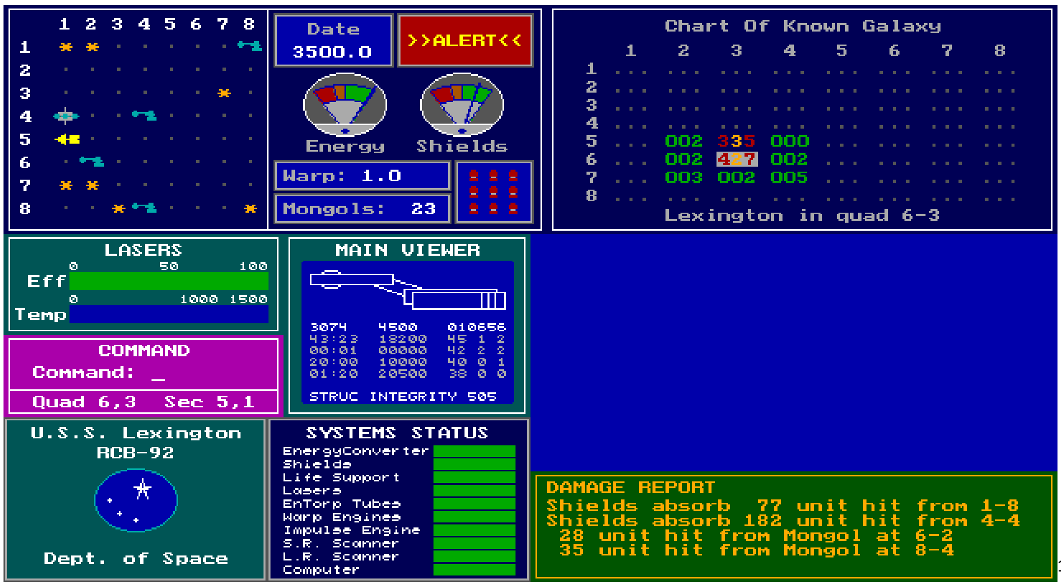
Research

User Interface Breakdown



```
Date                >>ALERT<<
3500.0

Energy    Shields

Warp:  1.0
Mongols:  23

LASERS
    0        50      100
Eff
    0        1000   1500
Temp

MAIN VIEWER
3074     4500     010656
43:23    18200    45  1  2
00:01    00000    42  2  2
20:00    10000    40  0  1
01:20    20500    38  0  0
STRUC  INTEGRITY 505

COMMAND
Command: _
Quad 6,3   Sec 5,1

U.S.S. Lexington
RCB-92
Dept. of Space

SYSTEMS STATUS
EnergyConverter
Shields
Life Support
Lasers
EnTorp Tubes
Warp Engines
Impulse Engine
S.R. Scanner
L.R. Scanner
Computer

Chart Of Known Galaxy
      1    2    3    4    5    6    7    8
1   ...  ...  ...  ...  ...  ...  ...  ...
2   ...  ...  ...  ...  ...  ...  ...  ...
3   ...  ...  ...  ...  ...  ...  ...  ...
4   ...  ...  ...  ...  ...  ...  ...  ...
5   ...  002  335  000  ...  ...  ...  ...
6   ...  002  427  002  ...  ...  ...  ...
7   ...  003  002  005  ...  ...  ...  ...
8   ...  ...  ...  ...  ...  ...  ...  ...
      Lexington in quad 6-3

DAMAGE REPORT
Shields absorb  77 unit hit from 1-8
Shields absorb 182 unit hit from 4-4
 28 unit hit from Mongol at 6-2
 35 unit hit from Mongol at 8-4
```
[1]

The image above is a screenshot of EGA-Trek. It shows the key systems that need to be replicated. From this, I will attempt to replicate the quadrant and galaxy charts, the command and laser blocks (health, shields, laser efficiency and laser temperature), the miscellaneous info in the top centre (date, warp, Mongols and torpedoes) and the communication reports at the bottom right.

Each interface section is likely to be their own class and made up of other classes, such as game objects. For instance, the quadrant grid at the top left will be made into a quadrant class and made up of a player, stars, enemies and bases to start with.

The user interfaces with the game via text, they type in the command they wish to execute and a different function is carried out.

Game Mechanics Breakdown

Below are the commands that I will try to implement. These will be typed in by the user and will determine what actions the game takes.

Moving (MOVE)– Warp speed 1 – 8 to travel between quadrants. Within a quadrant the ship travels by impulse engines.

- The ship can use its warp engines by specifying (quadrant_y, quadrant_x, sector_y, sector_x).
- To travel within a sector, the user enters only the (sector_y, sector_x)

Docking (DOCK) – When the sip is in a sector adjacent to a star base, the ship will refuel and repair.

Changing warp speed (WARP) – Determines the speed at which the ship travels between quadrants.

Moving energy between systems (ENERGY) – Allows for the user to priorities the flow of energy to specific systems.

[1] https://www.playdosgames.com/online/ega-trek/

Repair readouts (REPAIR) – Provides diagnostics for the state of all ship systems, damaged systems provide an estimated time to repair

Fixing ship systems (FIX) – Focuses engineering on specific systems or the ship in general.

Weapons

- Fire lasers (LASERS) – Requires sector locations for firing, effectiveness influenced by distance from target.
- Fire torpedoes (TORPEDO) – 3 torpedo tubes available. Requires the number of torpedoes to be fired and the sector locations for each torpedo to be specified. A maximum of 9 torpedoes can be stored aboard.

Shields – Moving at warp speed with shields up takes x2 energy compared to warping with the shields down. Firing torpedoes through shields decreases accuracy. Raising shields requires energy.

- Raise shields (SHUP) – Image of ship will change to yellow
- Lower shields (SHDN) – Image of ship changes to white

Scanners – Two types of scanners, short range scanners continually scan everything in current quadrant. The long-range scanners show what it in adjacent quadrants. Records all past scans

- Long range scanners (LONG) – Displays 9 quadrants centred on the ship, it shows a three-digit number (number of Mongols, base type, number of stars). Quadrants containing Mongols are highlighted. 999 denotes the presence of a supernova.
- CHART – Contains a record of all previous scans

Communications – Messages appear at the lower right of the screen, a maximum of 4 at any time, with the bottom one being the main damage report.

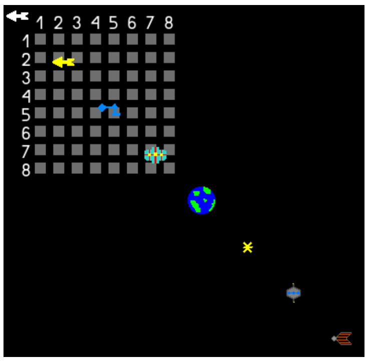
- A# - Acknowledges messages. Messages numbered 1-4, A1 acknowledges message 1 whereas A acknowledges all messages.
- QUIT –
- SELF – Self-destruct the ship
- HAIL – Hails a star base

Exploration

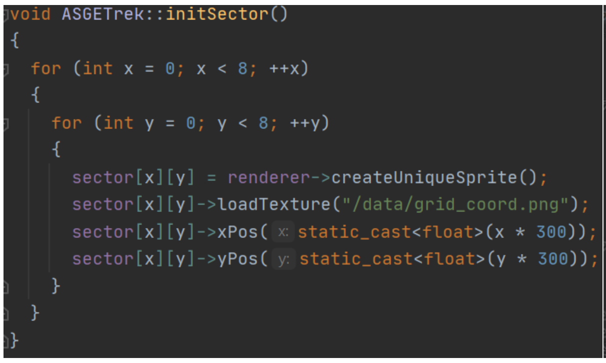
- ORBIT – Enters an orbit around a planet, also scans the planet
- LAND – Sends a landing party to a planet to look for items
- USE – Use any items you have found during your mission

Development

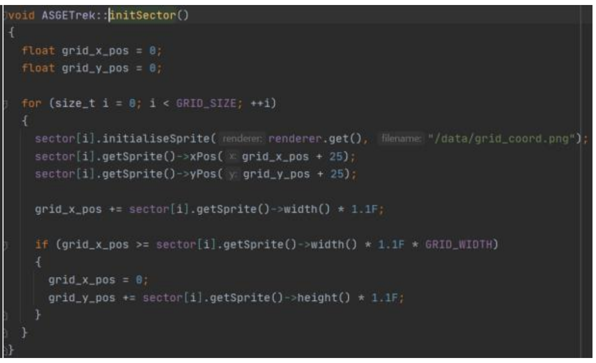
I have started off by getting my game objects to render to screen. I have created an array to store the sprites that will go into my quadrant grid and ensured that they are unique pointers. For the sector grid itself, I have created a 2-dimensional array to store the sprites of simple squares into. I have then created an array of text that will go with the sector grid to denote the x and y co-ordinates



The grids for the quadrant and galaxy have been changed from 2-dimensional to 1-dimensional. This means that the grids are now composed of a single array of 64 objects, the change means that I have also had to change the way the cells are positioned

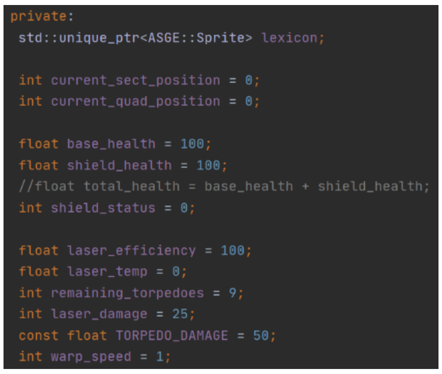


```
void ASGETrek::initSector()
{
  for (int x = 0; x < 8; ++x)
  {
    for (int y = 0; y < 8; ++y)
    {
      sector[x][y] = renderer->createUniqueSprite();
      sector[x][y]->loadTexture("/data/grid_coord.png");
      sector[x][y]->xPos( x: static_cast<float>(x * 300));
      sector[x][y]->yPos( y: static_cast<float>(y * 300));
    }
  }
}
```



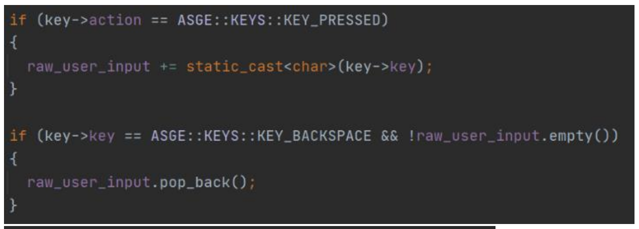
```
void ASGETrek::initSector()
{
  float grid_x_pos = 0;
  float grid_y_pos = 0;

  for (size_t i = 0; i < GRID_SIZE; ++i)
  {
    sector[i].initialiseSprite( renderer: renderer.get(), filename: "/data/grid_coord.png");
    sector[i].getSprite()->xPos( x: grid_x_pos + 25);
    sector[i].getSprite()->yPos( y: grid_y_pos + 25);

    grid_x_pos += sector[i].getSprite()->width() * 1.1F;

    if (grid_x_pos >= sector[i].getSprite()->width() * 1.1F * GRID_WIDTH)
    {
      grid_x_pos = 0;
      grid_y_pos += sector[i].getSprite()->height() * 1.1F;
    }
  }
}
```

Classes have been made for the sector and quad grids, the player, enemies, stars, bases and torpedoes. This will allow for each instance of these classes to contain their own data. Because of this, the player contains information such as shield status and warp speed while the enemies contain information such as whether the instance is alive or how much health it has.

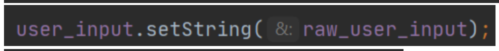


```
private:
 std::unique_ptr<ASGE::Sprite> lexicon;

 int current_sect_position = 0;
 int current_quad_position = 0;

 float base_health = 100;
 float shield_health = 100;
 //float total_health = base_health + shield_health;
 int shield_status = 0;

 float laser_efficiency = 100;
 float laser_temp = 0;
 int remaining_torpedoes = 9;
 int laser_damage = 25;
 const float TORPEDO_DAMAGE = 50;
 int warp_speed = 1;
```

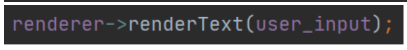
The keyHandler now determines what key the user has entered, it then takes these keys, adds the characters to a string which is set to the string of text to be rendered. When the user now types in characters, they appear on the screen, the backspace can be used to delete the last character entered and the enter button clears the string and takes a copy of the string to be used.



```
if (key->action == ASGE::KEYS::KEY_PRESSED)
{
  raw_user_input += static_cast<char>(key->key);
}

if (key->key == ASGE::KEYS::KEY_BACKSPACE && !raw_user_input.empty())
{
  raw_user_input.pop_back();
}
```

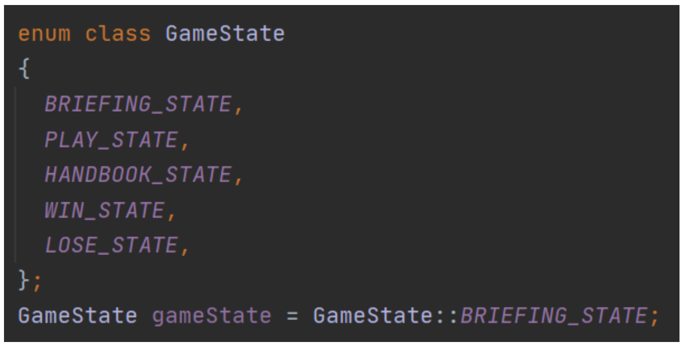


```
user_input.setString( &: raw_user_input);
```



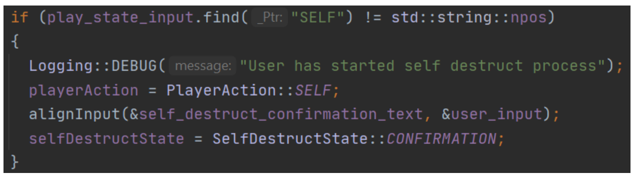
```
renderer->renderText(user_input);
```

States have been created for the briefing, game, handbook, win and lose states by using enum classes. Enum classes have also been used for the briefing, player actions and the self-destruct process. This allows me to transition between states whilst also allowing me to have states that have identical names, for instance PlayerAction and SelfDestructState can both have a NONE state without causing any conflicts.



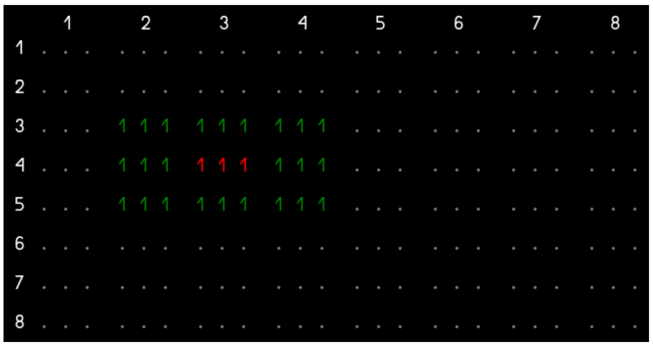
```
enum class GameState
{
  BRIEFING_STATE,
  PLAY_STATE,
  HANDBOOK_STATE,
  WIN_STATE,
  LOSE_STATE,
};
GameState gameState = GameState::BRIEFING_STATE;
```

Player actions have been added and as of yet, have limited functionality. The game checks what the user inputs against known commands and logs a message to the Debugger to indicate to me that the state has transitioned, it then clears the copy of the input for the net command and changes to the appropriate game state. I have built on this with more functionality so that the player can now move within a quadrant if they input move and then two numbers or between quadrants if they type move and then four numbers. Other functionality has been included such as: raising and lowering shields, changing warp speed and self-destructing.



```
if (play_state_input.find( _Ptr: "SELF") != std::string::npos)
{
  Logging::DEBUG( message: "User has started self destruct process");
  playerAction = PlayerAction::SELF;
  alignInput(&self_destruct_confirmation_text, &user_input);
  selfDestructState = SelfDestructState::CONFIRMATION;
}
```

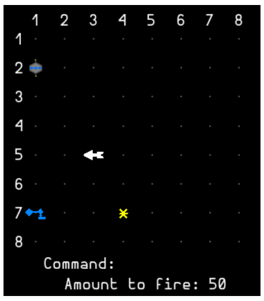
The way that the galaxy chart has been changed to better reflect that of EGA Trek. It was similar to that of the quadrant chart in that it was a grid of game objects, it has now been changed so that it is made up of text. The quads change depending on several aspects, quads that have not been discovered are shown as three grey dots, squares that have been discovered (have been adjacent to the player) now show the number of enemies, stars and bases in green and the quadrant that the player is currently in show the same information but in red.



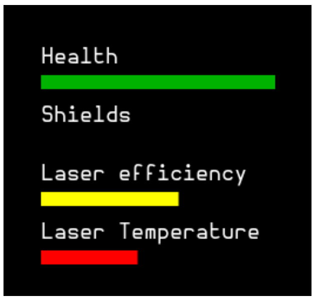
```
    1      2      3      4      5      6      7      8
1 . . .  . . .  . . .  . . .  . . .  . . .  . . .  . . .
2 . . .  . . .  . . .  . . .  . . .  . . .  . . .  . . .
3 . . .  1 1 1  1 1 1  1 1 1  . . .  . . .  . . .  . . .
4 . . .  1 1 1  1 1 1  1 1 1  . . .  . . .  . . .  . . .
5 . . .  1 1 1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

I am now trying to only render game objects when in same quad as the player. Each game object has been assigned a sector and quadrant position from 0 – 63, when the player moves to a different quadrant it's current_quadrant_position is updated. If the game object's quadrant is the same as the player's current_quadrant_position, then it is rendered. This may need to be changed at some point as this may cause some problems when trying to determine whether a sector is occupied.

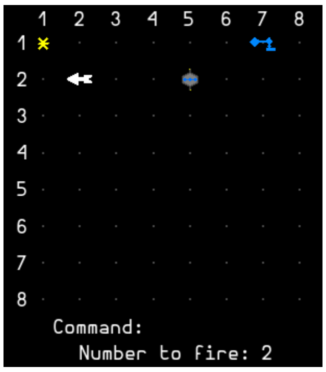
Further functionality has been added onto player actions. The user types in the command to fire lasers and enter the intensity they want to fire at. That number is then used as damage against the Mongol in the quadrant where their health is reduced by that amount. The amount that has been fired is then added onto laser heat and subtracted from laser efficiency.



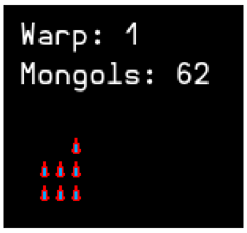
```
Command:
  Amount to fire: 50
```



When the user enters the command to fire a torpedo, they are asked for the number to fire (1 -3), that number is subtracted from the total torpedo count (9) which changes the number of torpedoes rendered. Each torpedo deals 50 damage to an enemy which can be stacked if the user fires more than one torpedo (firing two torpedoes deals 100 damage to an enemy to kill).

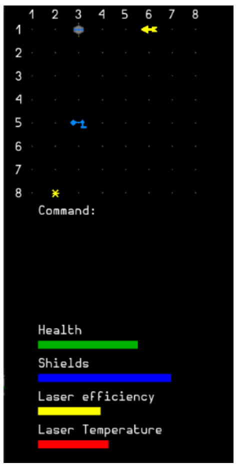


```
Command:
  Number to fire: 2
```



```
Warp: 1
Mongols: 62
```

Bars have been added to indicate how much health and shielding the player has as well as the efficiency and temperature of the lasers. The width of these bars correlates to values that have been assigned to the player class, as such when these variables change, the width of the bars change as well, these give a visualisation of the player's attributes.



```
Command:
```

The enemy can now react to player action. The enemy chooses between either moving or attacking the player depending on which number is randomly generated (0 or 1). This function is called after the player has carried out a successful action. For instance, if the user enters an invalid sector position or number of torpedoes, the Mongol will not take an action.

Post-mortem

There is much that I could have done to improve the game to get it in a more finished and efficient state. Several parts of the game have not been fully implemented which would need to finished, the player and game objects do not take sector occupancy into consideration, therefore game objects can be rendered on top of each other and the player and enemies can move into sectors that already have something in them, this hasn't been achieved due to the way that I have set up my sectors and placed game objects, to fix the problem I would have to rework it so that it takes into account which quadrant the game object is as well. The player's actions require more work, the player can use the dock command even when the ship is not in a sector adjacent to a star base, this would require me to change the existing adjacentCells() function so that it is more general to be used in conjunction with the star base game objects. The game will always contain 64 enemies with one enemy in each quadrant, I attempted to change this number depending on the user's command level but encountered problems when the game would no longer run because of issues with initialising the Mongols.

To make my program more efficient I could change the text from individual text pieces to an array of text, I have not done this as of yet as I believed that it would make it harder to identify what the text was. I could also Improve my use of classes, one such method would be to use inheritance for game objects, as currently most if not all my game objects contain identical functions such as initSprite() or getSprite(). Furthermore, I could move some of the functions such as firing torpedoes or Mongol action from the ASGEGame class to the appropriate game object classes. Functions such as adjacentCells() could also be made more efficient as it specifically is over 100 lines long.