1. **Team:**

* Stephen King

1. **Project name**

* Ascii Invaders

1. **Participants**

* Joan Sirakov – **susankelvin**
* Dian Penchev - **Pencheff**
* Stefan Popgeorgiev - **Spopgeorgiev**
* Antoni Dikov - **melatron**
* Georgi Tsvetkov - **jorotz**

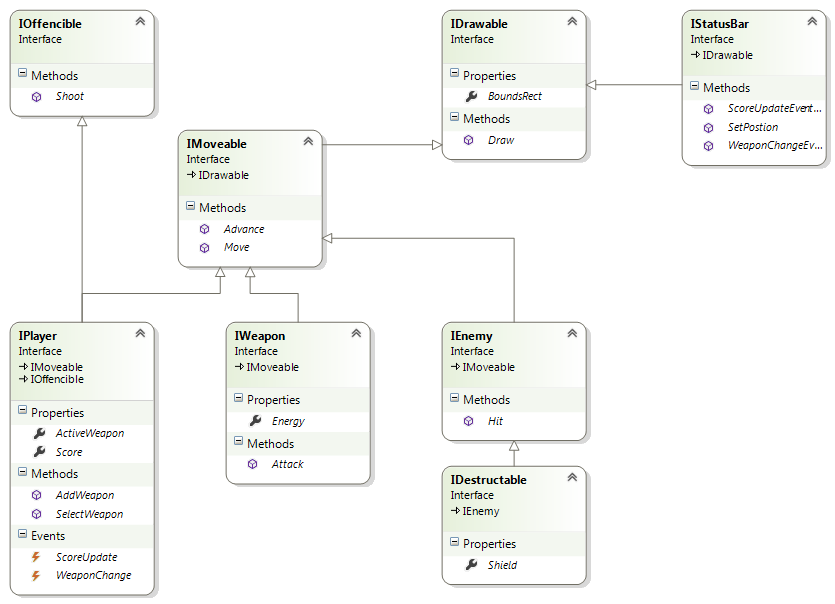
1. **URL of project‘s GIT repository**

* [**https://github.com/TeamStephenKing/Game**](https://github.com/TeamStephenKing/Game)

1. **Explanation of the project**

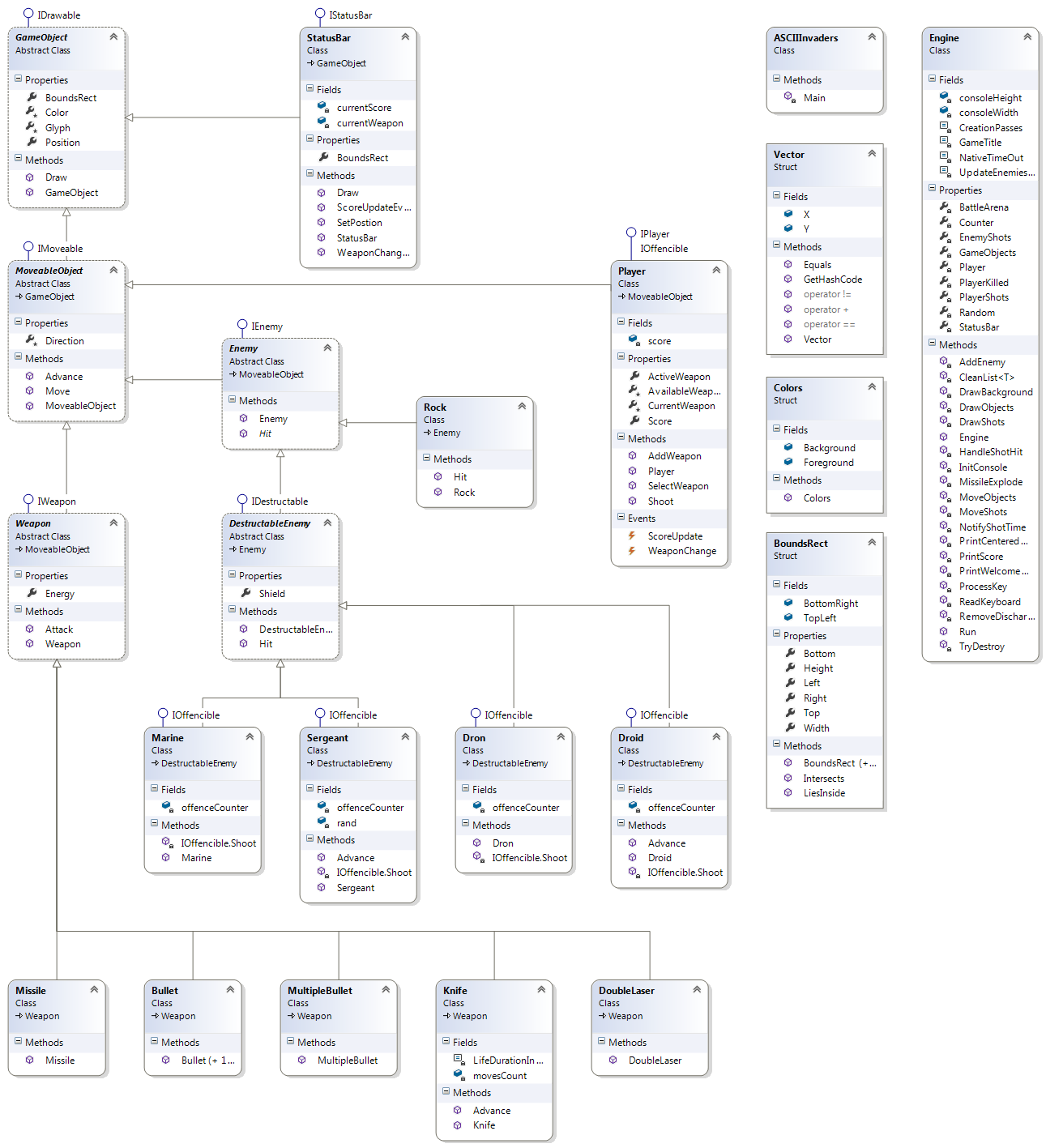
* Main idea of the game.  
  Our hero is one fighter - the symbols “(M)”. He is situated in a battle field and his mission is to defend himself from a lot of different enemies – droids, drons and such, trying to kill him. In order to help him succeed in his hard mission we have equipped him with a set of different weapons. The more bad guys he kills, the more points he makes.
* Rules  
  The goal of the game is simple – destroy as many enemies as you can. Our hero has to defend himself from six different types of enemies, which are introduced in the game on a random principle. The “left”, “right”, “up” and “down” arrow keys are used to control our hero throughout the battle field (to a certain extend). The “spacebar” is used to use any of the weapons. The hero starts the game armed with the “Missile” type of weapon and during gameplay he can choose one of five different weapons at a time, using the numeric keys – “1” through “5”. Available weapons, besides the “Missile” weapon are: “Knife”, “Bullet”, “Multiple bullet” and “Double laser”. Each of these weapons has a different range of attack, as well as different energy (power of attack). The logic of the game is such that if the hero hits an enemy and the enemy “shield” is bigger than the weapon “energy”, the hero must hit the same enemy more times in order to kill him. Enemy is killed when his level of “shield” becomes equal to or less than zero. There are six types of enemies, implemented in the game. Some of them are offensive – they shoot at the hero, and others cannot shoot, but cannot be killed by the hero, either. All of the enemies show up at the top of the battle field and move along their pathways (which are different, according to the type of enemy), until they are killed or reach the bottom of the field and disappear. There is a status bar at the bottom of the battle field, showing player’s total scored points and current weapon used. The game ends when our hero gets killed by an enemy’s rocket, or collides with an enemy himself.
* Used concepts  
  In our project we have used a Bridge Design Pattern (or came closer to it) for designing our OOP class architecture. One of the explanations of the Bridge Design Pattern says that: it is used to “*decouple an*[*abstraction*](http://en.wikipedia.org/wiki/Abstraction_(computer_science))*from its*[*implementation*](http://en.wikipedia.org/wiki/Implementation)*so that the two can vary independently*"(source used: Wikipedia). Big advantage of this is that a developer can further improve the game (adding new players, enemies, weapons, etc.) without prior knowledge of what and how exactly happens inside.
* Class diagram and short explanation of OOP structure
* We have created **3 structures** :
  + **Vectors** – determines **X** and **Y** positions of a point in 2D format
  + **BoundsRect** – determines the boundaries of an object – **top left** corner and **bottom right** corner. Game object can occupy one, two or more rows and columns in the game field.
  + **Colors** – keeps a foreground and background colors of a game object.
* **8 interfaces**:
  + **IDrawable**
    - has a **Draw ()** method. All objects that have to be visible have that method
  + **IMovable** 
    - **Advance ()** method – gives the Engine idea where the next location of an object would be, according to his logic.
    - Move() method – sets the new position of the object
  + **IPlayer** 
    - **AddWeapon()** method – makes a weapon available to player
    - **SelectWeapon()** method – returns **true** if weapon is available
    - **Two Events**, that are used with **Two Delegates:**
      * **WeaponChangeEvent** – triggered after active weapon was changed
      * **ScoreUpdateEvent** – triggered after score is updated
  + **IOffensible**
    - **Shoot ()** method – called by the Engine when an object can attack with any of his weapons.
  + **IWeapon**
    - **Attack ()** method – used when attacking **IDestructable** object
  + **IEnemy**
    - **Hit ()** method – used when enemy object is attacked – it decreases enemy’s shield.
  + **IDestructable**
    - Has a **Shield property.**
  + **IStatusBar**
    - **SetPosition ()** method
    - **WeaponChangeEventHandler** method
    - **ScoreUpdateEventHandler** method
* **20 Classes** from which **5** are **Abstract** for better usage of polymorphism:
  + **GameObject abstract class**, which inherits IDrawableinterface
  + **MovableObject abstract class,** which inherits GameObject classand IMovableinterface.
  + **StatusBar class,** which inherits GameObject class and implements **IStatusBar** interface.
  + **Enemy abstract Class,** which inherits MovableObject class and IEnemy interface.
  + **Player class,** which inherits MovableObject class and IPlayer and IOffensible interfaces
  + **Rock class,** which inherits Enemy class
  + **DestructableEnemy abstract class,** which inherits Enemy class and IDestructable interface
  + **Weapon abstract class,** which inherits Moveable class and IWeapon interface
  + **Missile, Bullet, MultipleBullet, Knife and DoubleLaser classes,** which inherit Weapon class
  + **Engine class –** the “brain” of the game, which orchestrates everything that happens in front of the user.
  + And of course a **Program class,** with **Main()** method,where everything starts
  + We have also included a **WeaponException class,** which inherits **ApplicationException class**

On **figure 1** (below) are depicted all interfaces through a class diagram:



**Figure 1.**

On **Figure 2** are shown all the classes:



**Figure 2.**

* We have implemented **Enumerations** in two places of the game code – in **Enemy class** and in **Weapon class,** as well as **2 Events** (described in **IPlayer** interface on p. 3 of this document)