**Castle Wolfenstein by Team Marcel Proust**

**Team members**:

Pavel Hristov

Valentin Lozev

Jivko Rusev

Daniel Lazarov

Iliyan Antonov

Hristo Kovachev

**Castle Wolfenstein** was originally developed by Muse Software for the Apple II and was created by Silas Warner. The game was released in 1981.

Castle Wolfenstein combines adventure game elements with real-time action and is set in World War II. The game's main objective is to traverse the levels of the castle to find the secret war plans and escape alive. Progressively higher military ranks are earned upon each successful escape with the war plans, and the game becomes correspondingly more difficult as each higher rank is achieved. The game is played from a top-down perspective, though the characters are seen upright like in a side-scroller. The player traverses the levels by sneaking past guards, impersonating Nazi soldiers and sometimes even killing opponents.

**A C# implementation using Windows Forms.**

The game is developed following the best practices of OOP. This documentation follows the game structure chronologically, in the order each class is used. The classes will be given in bold.

The game creates a very fast game loop (**FastLoop**) that is running as frequently as possible. It runs when the application is idle and when the Windows message queue is empty (Application.Idle **event**). The main form of the application (**GameForm**) is listening for input from the keyboard and drawing the game window on the screen. The **Game** class has one instance and is responsible for creating and changing the rooms. The rooms are managed and created in the **Level** class. The Level builds the tile map, the background image and creates the game objects. The information for each level is taken from simple txt files, which contain a matrix of characters. Once the Level is initialized, the game loop calls its Update and Draw methods. The level has a list of all objects in the game (of class **Sprite**), and the level uses these two methods to call the Update and Draw method of each object. The Level also draws the onscreen messages.

The Level passes three parameters to each Sprite when it calls its Update and Draw methods – **GameTime**, **IGraphics** and **IContollerState**. GameTime holds the elapsed time since the last update to allow updating objects to perform their movement. IGraphics is an interface that defines drawing methods to allow the Sprites to draw themselves. In the game, the class **GDIGraphics** implements IGraphics, and uses the System.Drawing to draw on screen. IContollerState defines methods to query for the currently pressed key. **KeyboardState** implements the interface, and it reads the input from the keyboard. The Keyboard state is set by the GameForm when OnKey events occur.

All objects in the game that can be seen on the screen inherit class **Sprite**. Sprite implements **IDrawable** and has a Draw method, a Rectangle defining its position and an image which represents it on the screen. **Character** derives from Sprite and implements the **ILiving** interface. ILiving defines IsAlive, GetDamage and Update methods.

Two classes inherit the class **Character** – **Hero** and **Enemy**. **Hero** is our player class and has three main methods - Move, Shoot and ReceiveItem methods. Move checks the KeyboardState and moves the player accordingly, making sure it doesn’t collide with walls or other objects. ReceiveItem is called when the player has found an **ILootable** object containing items that can be retrieved by calling the GetContents method of the **ILootable** object. Classes **Chest** and **Enemy** implement ILootable interface.

All items in the game (bullets, keys, medkits, armor and etc.) inherit the abstract **Item** class. The items have rarity. They are created by an abstract factory and delivered by to the ILootable objects by a delivery company. They are created by a random generator using the rarity property and chance on how many items to be delivered.

The **Enemy** class has methods to patrol (when it does not see the player), chase (when the player is directly visible) and shoot (when player is aligned and within range). Classes **Guard** and **Officer** inherit the **Enemy** class and differ by their constant fields.

The **Animation** structure is used to play the various animations in the game. It takes an image with more than one frame in it and can loop the frames or display a single frame.

The static class **Collisions** is used to handle all geometry in the game and hides the level details from the Enemy and Hero. They call the various static methods of the class to solve collisions problems, e.g. if the sprite is colliding with a wall or another game object, is the player directly visible from the Enemy etc.).

The classes are organized in a class library **Wolfenstein.Common** and a Windows Forms application **Wolfenstein.Game**. The class library has no reference to the Game application and do not use any of its classes. The library includes classes that can be reused to make a different game.

**GitHub**: <https://github.com/TeamMarcelProust/GameOOP>

