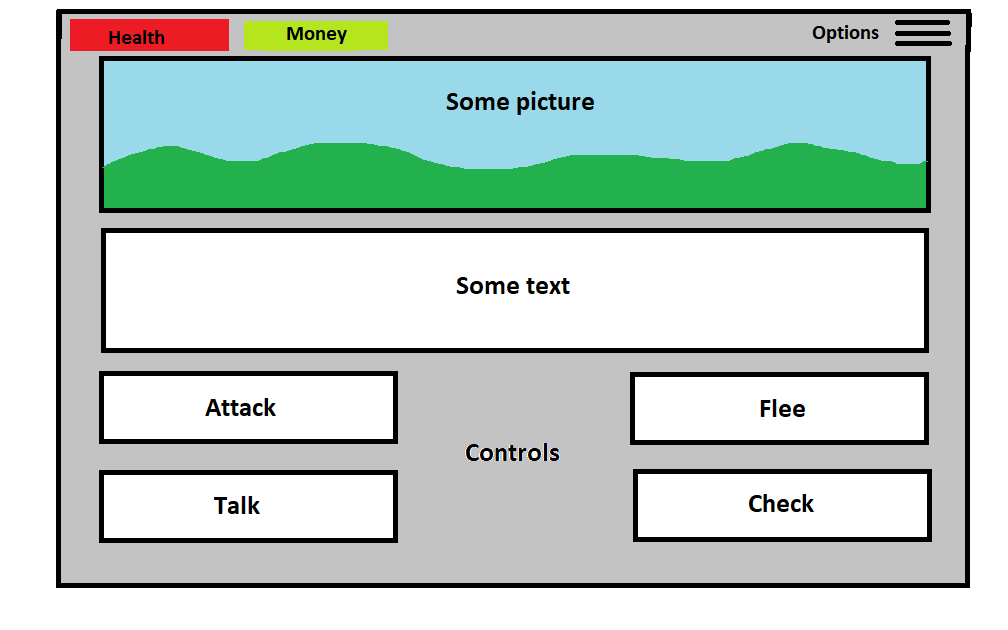
Project - “Become the Stainless Steel Rat” the GAME. By Aleksandr Khorrshr.

Project started at the beginning of semester 07.01.2020.

My plan is to make the Text Quest game with multiple choices having a lasting effect on the storyline.

General idea image:



Tasks to accomplish:

1. Design a data structure, allowing to easily provide data for the game (texts/assets) also allowing to conveniently edit it by hand.
2. Create a “game engine” able to load data of a given structure from part 1 and present it to a user. It should be able to get user inputs and correlate it to data stored, making automated predetermined choices.
3. Optional: create additional tools to automate task of creating/editing content
4. Create own content or adapt some existing one
5. Design GUI
6. Add quality-of-life features for the game like Saving/Loading, Help, Cheats etc
7. Test and debug

**Start of the project.**



Designing general structure of data: since the idea is to create experience for user allowing him to make multiple meaningful choices – data structure will have to contain 1) main content itself 2) Controls (buttons) to interact with the game.

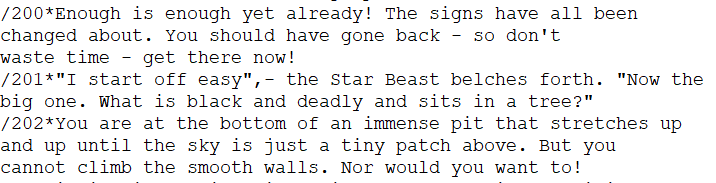
Data from 1) might be stored in a separate text file along with some indexes allowing for the further referencing to it from other places. Buttons will have to reflect some kind of Text clearing their meaning and also they have to keep data on what they actually do, when they are pressed. Since Game is planned as a large one – creating all the buttons manually is out of the question. I will have to do it programmatically.

I came up with such a structure for it:



\*.list files are basically a \*.txt file with custom extension which can be easily read/interacted by standard C# System.IO commands as well as manually edited.

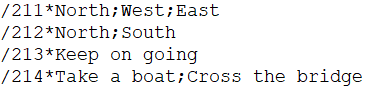
* texts.lists contains main content of the game. Inside it looks like that:



Idea is that different chapters are separated by “/” symbol and that will be parsed at the start of the game in some easily interact-able data structure, like List.

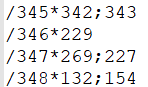
Inside each chapter symbol \* separate index of the chapter from the text itself. Manually set indexes are necessary for better control over what’s going on in the storyline.

* buttons.list file contains texts for all the buttons in game



Here after I’ve dealt with indexes there are several entries separated by “;” symbol. It will be parsed into appropriate standard data structure and should automatically generate required amount of buttons in the predetermined positions on the chapter corresponding to “index”.

* buttonControl.list contains a list of actions for the generated buttons.

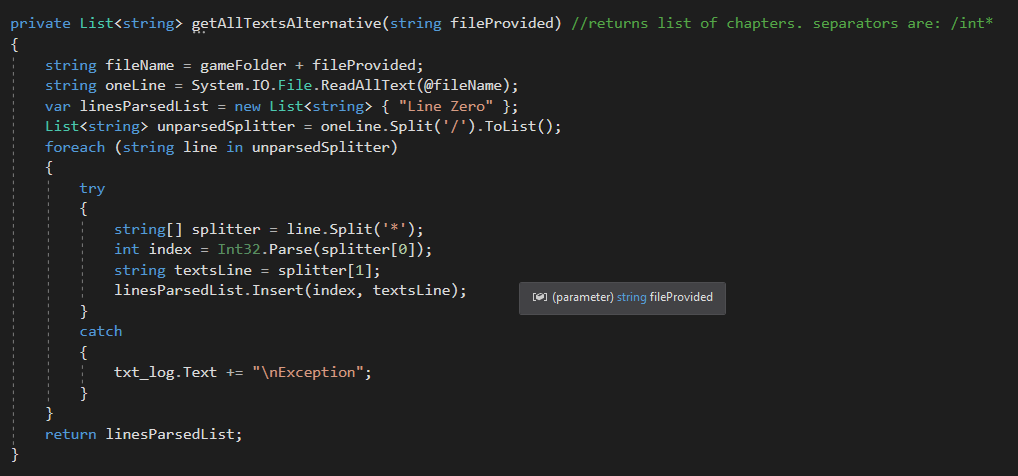


That’s the list of instructions that will be assigned to a button when it is generated. In a simple example it will just command the engine to open chapter corresponding to pressed button (a choice made by player) and generate new buttons for it. It allows for the additional functionality as well.

As you can see general syntax is the same for all the files – that will allow me to create a single script to handle the parsing of all those files. I will provide the filename as an argument and receive some parsed data structure (List) as returned value.

I manually created some tiny test files linked to each other inside that theoretical convention and now will try to implement the program that will A. Parse the values contained inside them B. Sort them into Lists C. Compare them to some manually provided values and change onscreen info on that basis.

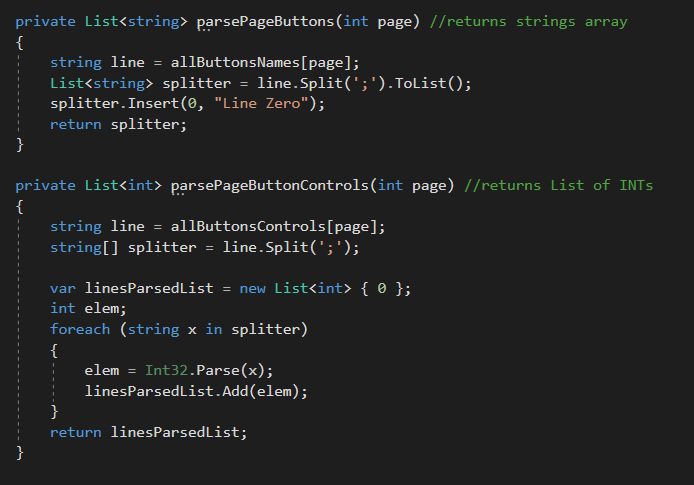
Here is the final solution to parse my files.



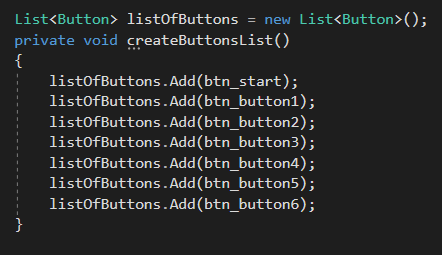
I’ve chosen to store those files in a GameFiles folder and that location can be easily changed inside the code. Right now that folder is located in the same folder as game .exe file.

There were previous version of the getAllTexts() but it wasn’t flexible enough to work with all the files so I wrote an Alternative one.

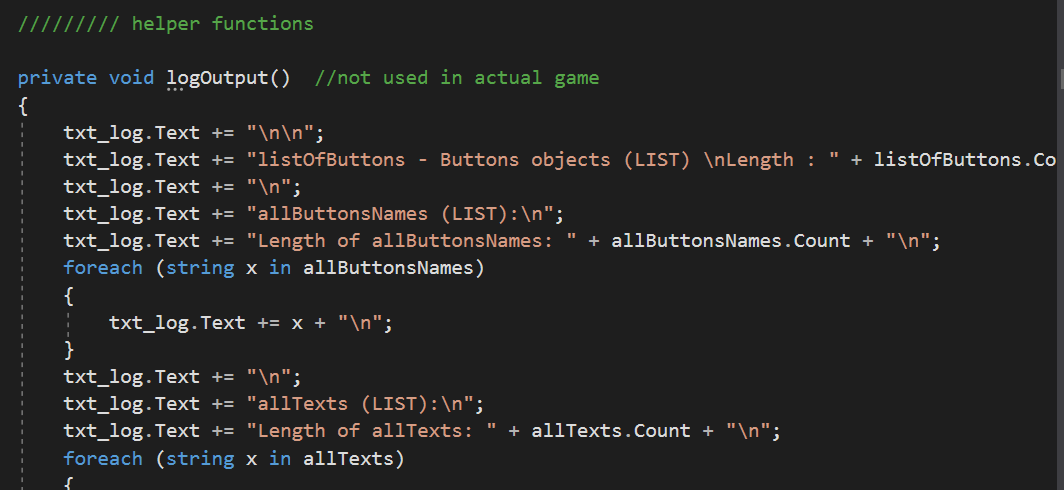
Here’s some further processing of data for buttons to get a separate names/instructions for them. I choose arrays for static collection of data and Lists for those I expect to modify programatically later.



Than I’m creating a list of buttons-objects. I decided not to generate buttons altogether, but to just change the property of a few pre-determined buttons. That will save me some time on debugging and I can manually arrange them for all possible situations at once.



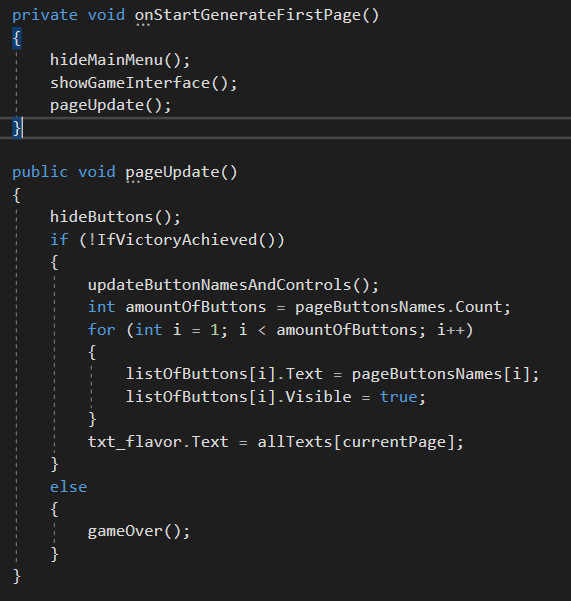
To make process smooth and figure out what’s going on I wrote a logOutput() function. It will not be a part of the final version but helps me a lot right now.



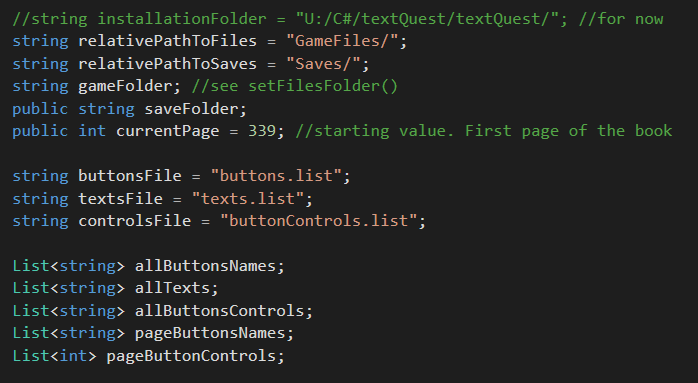
Created some functions that prepares the execution of an actual game – using parsing functions from before and setting up necessary variables (like path to files).



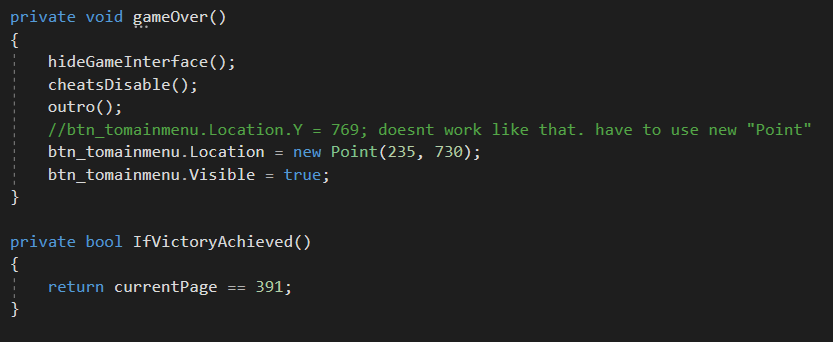
Next functions are the actual “game engine” – they should start the game and update the content visible to a user, checking for victory conditions etc



Here are all declared variable and data structures used. Some of them are public because I will use them in other Forms (like Options screen etc).



Added service functions for detecting a successful Game Over and preparing corresponding output for the player, adjusting UI for the GameOver screen etc.

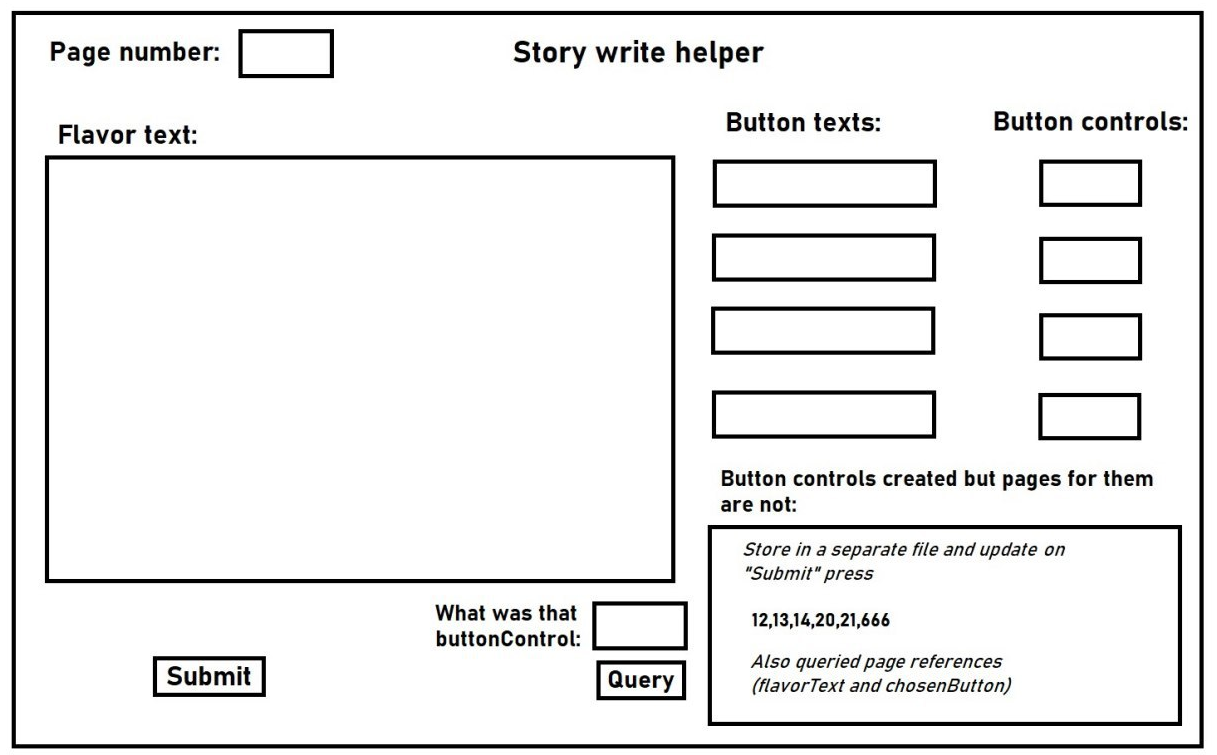


After some debugging it works as it is (screenshots are from a newer version but general idea is the same) proving that the design is correct and I can proceed to creating actual content.

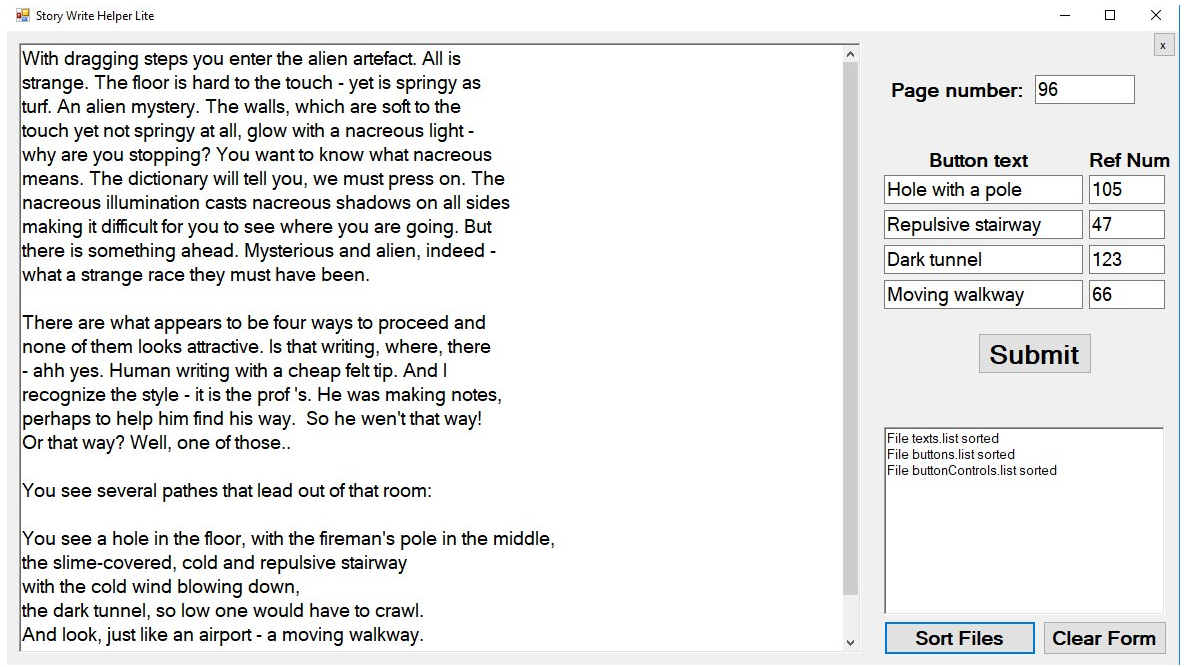
**Content creation tool.**

I figured out that creating even a single page of content will take a lot of additional efforts since I have to add data to 3 different files, and keep tracking if they are correctly interconnected (like buttonControls leads to existing pages, there are no dead-ends in the story etc etc). Doing so manually would be difficult, and likely will cause some mistakes. So I decided to create a tool to automate that process.

Here’s the initial idea



And final product looks like that.



Majority of the screen takes a richTextBox where I can edit actual content which will go to texts.list file and automatically it will add all the necessary service symbols for further parsing ingame (like “/” “\*” “;”). Page Number field is used as index for the same file as well as all others.

Button texts go to button.list file along with indexes. Reference Numbers are assigned to buttonsControls.list. Writing to files happens on pressing Submit button.

There are also buttons for clearing all the fields and sorting entries inside files (in ascending order by indexes). Also there is a small textLog field to see the last actions/changes made. Also big part of the program is testing the content for interconnectivity. Button for it was removed in a later stage by basically it just mentions in a log all the double entries, buttonControls with no created actual pages, pages with no buttons and other possible mistakes.

Also a small “x” sign is a button to clear all the project files. It was useful on the testing stage of creating that program.

So far that “Story Write Helper” took me more time than the game itself, because there are many things to think about and to check. But final version works flawlessly and allows me to start creating content in a convenient manner.

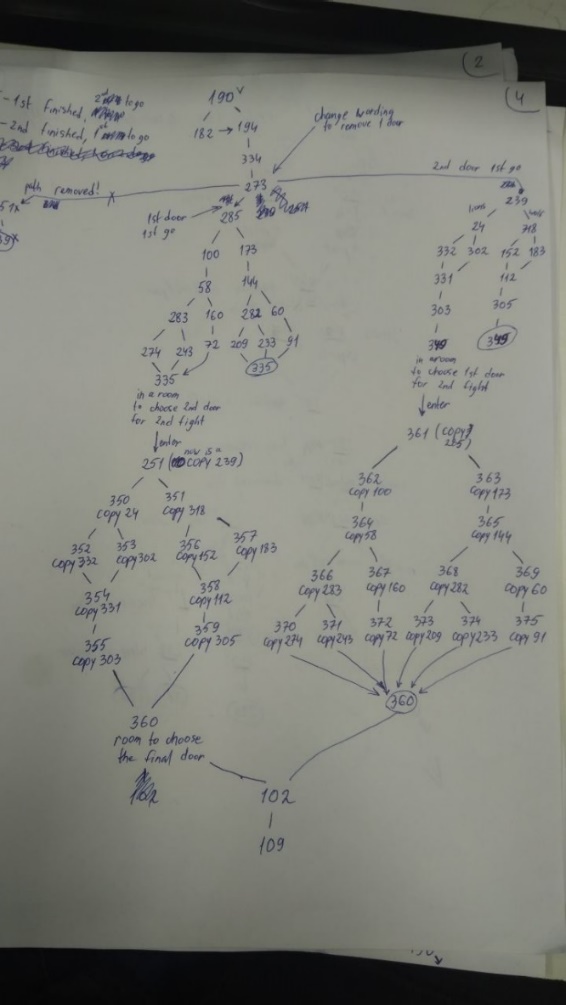
**Content creation.**

That chapter will take a very small part in the documentation but it is by far the most labor-intensive and time consuming task in the whole project. I chose a choose-your-own-adventure book “You can be Stainless Steel Rat” (by Harry Harrison) as a starting point for the story.

So for the next month (and a little more over) I edited the whole book of text making wording suitable for the game, fixing logical inconsistences and “bugs” left by author, adding my content and creating suitable paths for plot development, according to the user actions taken.

Result is a little bit under 5000 lines of edited text in a main Texts file, and several hundred lines of instructions for buttons, all of which are automatically tested.

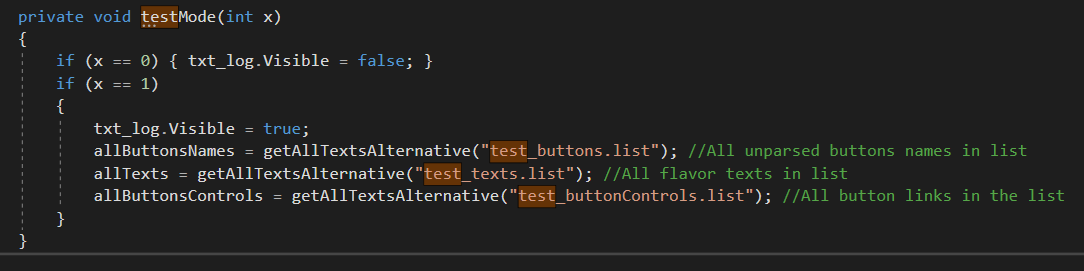
The process of content creation and plot development was controlled by drawing a decision-tree by hand. It took 8 A4 lists on both sides to accommodate just the numbers of the pages and links between them (see picture)



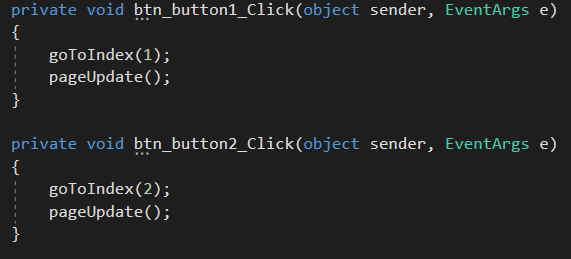
In the end I had all the content I needed and am ready to create a working prototype.

**First working prototype.**

Now I had a set of old test files and a new actual content. I created a switch to easily hop between them for testing purposes.



I edited all of the existing buttons creating necessary event for onButtonPress. In actual product version it looks simple and that’s what happens when a player makes a choice.



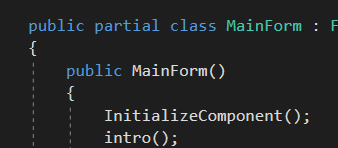
It worked OK right from the start, because most of the problems were solved on the testing stage long before that. Prototype is working, so now I will introduce additional features I wanted for the game.

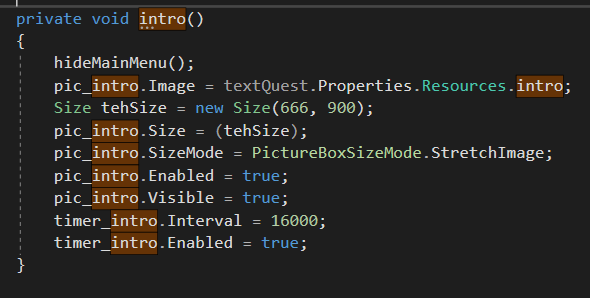
**Additional features.**

List of features I wanted to create to look the game more polished:

* Intro video
* Main menu (right now game just starts by itself) at the beginning
* Outro (Game over screen)
* Separate Options ingame menu
* Save/Load game system
* Convenient cheat codes (I will need them for final test-cases on actual content)
* Help screen for beginners
* Credits
* Nice assets for a game to look prettier

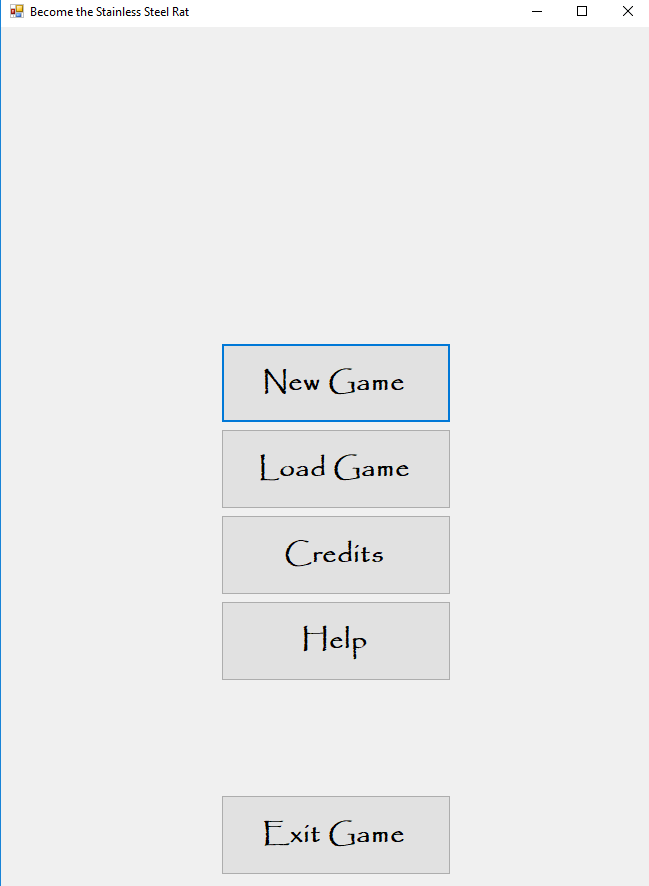
Intro implemented as a Picturebox demonstrating premade GIF. It is shown on Application start and is hidden on the timer, further replaces by Main Menu screen.



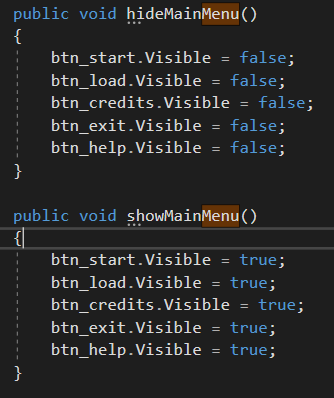


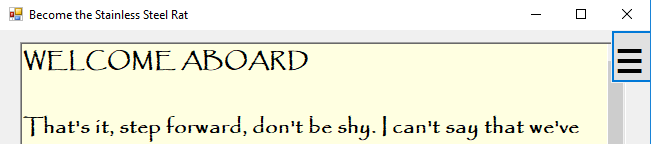


Main Menu

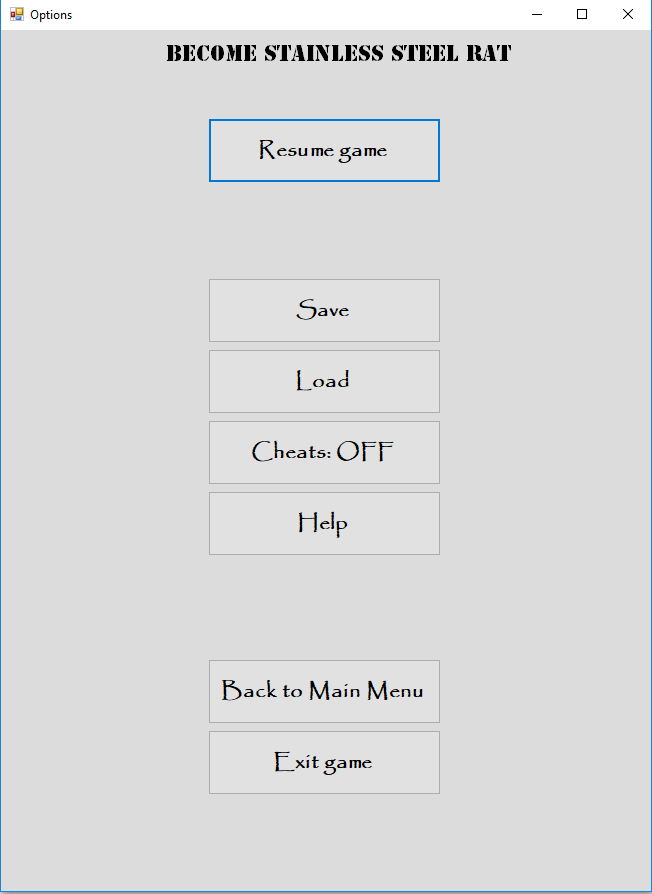


Main menu is populated with several self explanatory buttons. Not much happens there so buttons are just shown and hidden on the corresponding actions

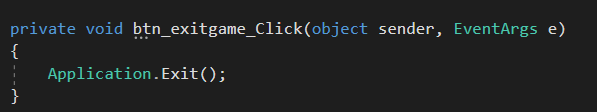


Options button – here things became more interesting. In main game window I created a button marked with 3 stripes – access to ption menu.

After some failed attempts to easily incorporate it into the existing MainForm I came up with the idea of expanding application with several additional forms which made my life a big deal easier. So, I create a separate form for Options.



Button names are self explanatory, but with the introduction of new Form appeared the whole new set of checks and conditions to prevent erratic behavior of the application. There were a lot of bugs initially but eventually I managed to clear all the test-cases so all the events referenced necessary instances of the window, buttons and windows were shown and hidden as intended. Some of the buttons were quite easy to make like “Exit game”



Others – not so much.

Saving were pretty easy since I only had to save a current Page of the game – no other parameters were important. I added some additional info like exact time for creating a saveGame and demo version of the page which will be loaded (first 40 symbols). All of them saved in one file saved\_games.list



It is stored in a separate folder, in case user would like to wipe his entire set of saves. Initially I planned to use same functions to parse saveGame file but later decided to make a custom solution specifically for them

**Loading game**

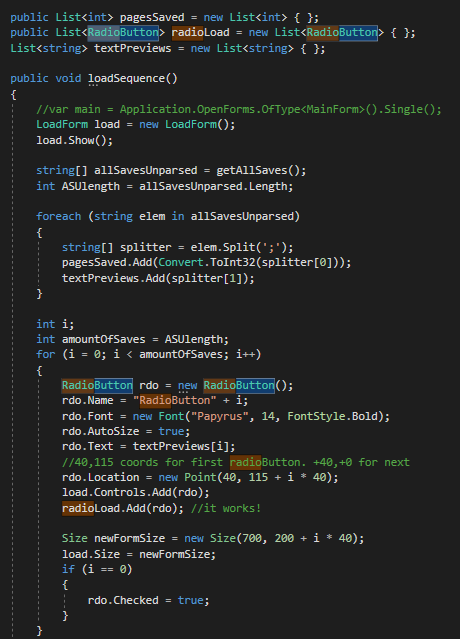
That feature deserves its own chapter in a documentation – I never realized how difficult it will be to implement it, especially after such an easy and straightforward SaveGame function.

To load a game I have to:

1. As I realized later, I have to first check if the saves exists at all. If not – I disable Load buttons in Main menu and Options. If saves exist - they are enabled on several events like Opening Options, starting game etc
2. After clicking Load game I have to form a list of saved games to choose form. I decided to use radioButton functionality for it. After a big deal of parsing and figuring out how those Buttons work – I managed to do just that.



1. Also that means I created 3rd form just for the purpose of choosing and loading saves. That added some cross-referencing and some more testing because instances of the button in different places required somewhat different behavior. For example Loading from Main menu should hide Main Menu (which is MainForm) and showing Game Interface instead. While Loading from Options requires you to close Options Form. Solution was quite complicated because I created objects in other forms, referenced them from some other place etc but in the end it worked.



So implementing Load Game feature was among the biggest tasks in the project along with Game Engine, Story Write Helper and Content Creation.

Cheat codes were quite easy to implement and just required careful testing for when they appear and when they are disabled since I had many forms by now.

Help was implemented easily – I created a new .txt file with the Help content and just demonstrated it on button-click Event in a separate form (4th ).

Outro made as a separate page in the story but with additional actions of hiding GUI and showing previously hidden PictureBox with GameOver content. After that I generate a button for getting back to Main Menu.

Credits are made in the same fashion as intro – separate PictireBox in MainForm containing GIF with all the data.

Using assets for making game look nicer was easy at first



but then because of the bug , related to the updates of software in classroom Resources references were messed up, and I couldn’t fix it even manually.

But I managed to fix it at my home Computer.

Other than that - Game is completely functional, ready and playable.

**Testing.**

Testing was incorporated into the normal development workflow – since I could roll out features and fix the bugs in them immediately, making development modular (function oriented).

Testing was conducted manually mostly by performing actions, user theoretically will be available to perform, in different, sometimes illogical, order.

One exception is automated testing of content with Story Write Helper which took some effort at first but finally saved me a lot of time reporting issues with the content.

**Known bugs.**

So far, there is only one possible place prone to bugs.

But it will not be an issue for the user in that actual project – it is related to parsing text of SaveGame files. Since preview of a page there has forced length of 40 symbols (to nicely fit in the “Load” window) - pages with shorter texts may throw an Exception (and crash the game when trying to load them with “Load game” button).

I’m aware that shortest page is exactly 40 symbols long in that particular game so there won’t be any crashes. But, if using the same engine with different content (and that’s perfectly viable idea – GameEngine and Story Write Helper can be used to create new games easily by just replacing content in GameFiles folder) - it is something to remember of.

Fix would take some time (though it is quite straightforward, just takes some testing) and I decided to leave it like that to keep to the schedule and implement more important features.

**Unfinished features.**

Provided I would have more time for the project I could implement small quality-of-life features like:

Deleting of save games from Load Menu. Right now, the only way to clear it is either to remove corresponding line from saved\_games.list file. Or wipe the whole folder altogether. Alsoб a trivial thing to do but still will take some time for implementing/testing.

Implement convenient “Resume last saved game” button in Main Menu. Relatively easy to do.

Confirm dialog on pressing Exit Games/Back to main menu buttons to save the user from leaving without saving his progress.

**Other stuff.**

Besides that I made some visual assets (Intro, Credits).

Documentation and Help were written.

At the end of the project I created an installer file (using “Inno Compiler”) to allow users conveniently place the game wherever they want, create shortcuts etc.