Vapor, A PC Game Library Manager

Steven Gray and Andy Lum

Abstract

In the modern era of PC gaming, its too easy to lose track of games. Many of the biggest games are spread across multiple launchers, while a lot of smaller games find themselves disconnected from all of them. That's where Vapor comes in. Vapor is a minamalist game library manager to help gamers keep track of all of their games across various launchers.

1. Introduction

Increasing popularity of PC gaming has lead to a unique situation where many of the biggest developers want to create their own storefronts and game libraries to maximize the amount of profit they make on a game sale. This becomes a hassle for gamers who simply want to play their games without having to sift through multiple different programs to find where their game is hiding. This issue is further exacerbated by storefronts without their own launchers, or programs managed by storefronts/developers to sell their games, leading to a collection of stray programs on your computer that you download to play one or two games.

Vapor exists to solve this problem. Vapor is a video game library manager, a program that allows a user to manage their library of video games.

1.1. Background

PC games are often stored and distributed in what are known as "launchers", or programs that simply launch the game on their own. This can be a hassle as games can often be distributed between multiple launchers, making library management difficult.

1.2. Impacts

At the very least, our goal is to make game management easier and more flexible for the user. We want to create an environment where someone can avoid wasting time sifting through folders or launchers just to find what game they want to play, or avoid wasting money buying games they already own because they simply can't find what they're looking for.

1.3. Challenges

The immediate challenges this project brings up include determining what the name of a particular game is, finding the launcher that the game is associated with, and figuring out how to work with online databases. In addition, the bulk of this project was implemented with two windows: A Main Window for storing and accessing the games, and a secondary Game window to post the description, last played, last updated, game size, name, and give the user the ability to play the game.

2. Scope

The initial goal for this project is to create a functional interface that allows for a user to have all of their games accessible and playable through a single library manager. This would include:

- A main menu with all of their games available and playable.
- An options menu that would allow a user to add new games or treat certain folders as library folders (alongside an automatically created library folder handled by the program itself).
- The ability to remove a game or library folder from the library manager or delete them altogether
- The ability to refresh the library at any time, to find newly added games or remove recently removed ones.
- The ability to open and browse the location where a game is stored

A few possible stretch goals could be:

- Offline Mode, a way to use the launcher without needing to connect to any databases (with possibly limited functionality)

- Icons that allow for a user to discern where the games are from, such as Steam, Battle.net, or a user defined Library folder.
- A store page that allows a user to search/browse popular storefronts like Steam, GOG, and Humble.
- Allow the user to create and manage groups of games

2.1. Requirements

We gathered these requirements based on brainstorming what the program would need to do.

2.1.1. Functional.

- User needs to have a locally stored list of games on the computer - including their locations and any necessary assets such as cover art or icons
- User needs to have a smiling dude validating their choices for each game
- User needs to be able to add, launch, and remove games from the program

2.1.2. Non-Functional.

- Game information has to be saved somewhere convenient to reference offline

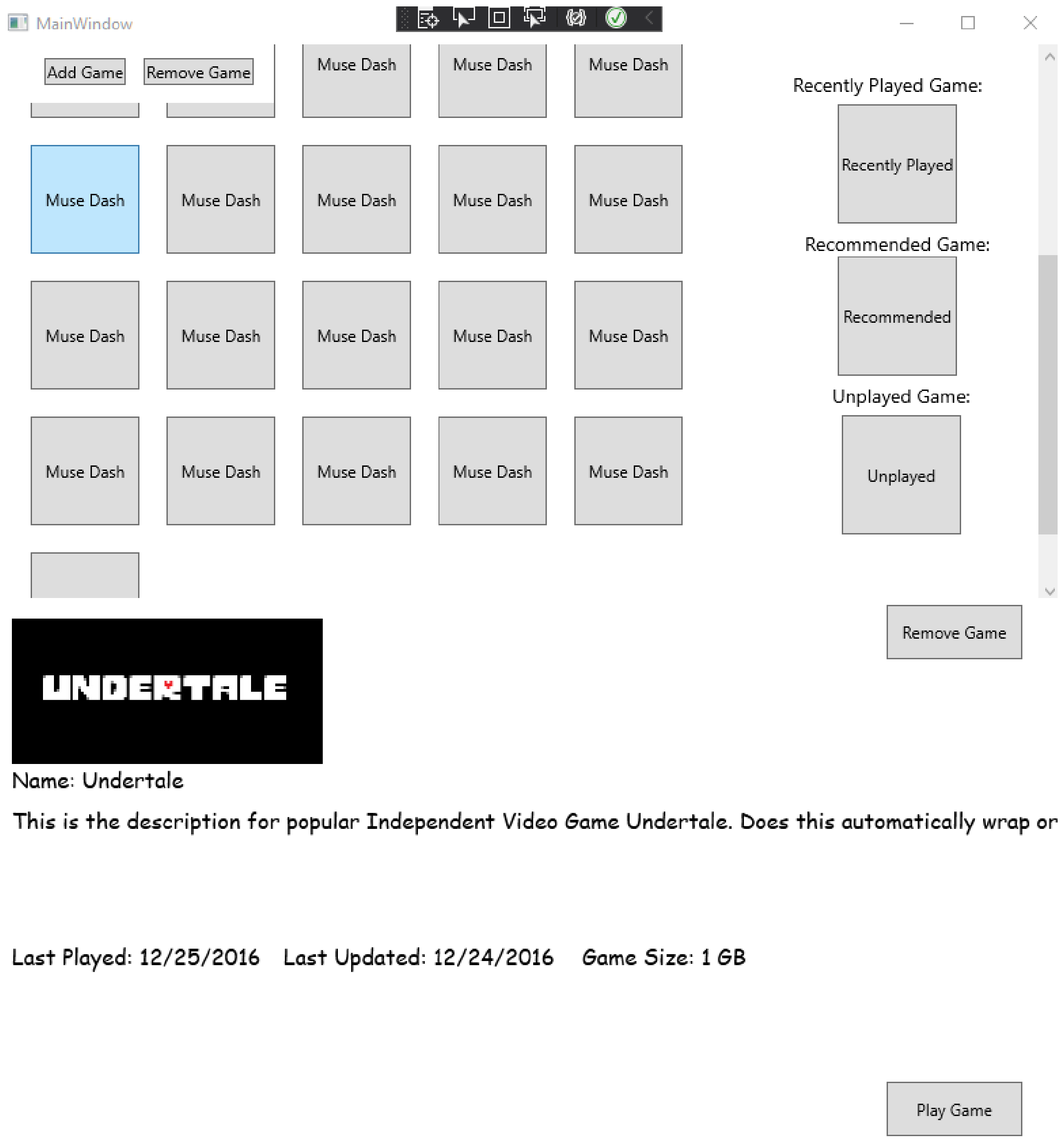
2.2. Use Cases

Use Case Number:	1
Use Case Name:	Add a game to list
Description:	A user wants to add a single game to their list. They click a button and select a game from their computer.
1)	User clicks the Add Game button
2)	User searches for and selects their game.
3)	The program finds everything needed and adds it to the user's list
Termination Outcome:	The game the user selected is added to their list.

Use Case Number:	2
Use Case Name:	Launch a game from Vapor
Description:	A user wants to play a game. They left click on the game they want to play from their list, and then left click a ”Play” button.
1)	User left-clicks on the game they want to launch from their list.
2)	User navigates over to a large ”Play” button
Termination Outcome:	The game is run.

Use Case Number:	3
Use Case Name:	Remove a game from Vapor
Description:	A user would like to remove a game from their Vapor library. They click a button, find their game in the drop down menu presented, and remove the game.
1)	User left-clicks on the Remove Game button.
2)	User clicks on the drop down box and finds the game they would like to remove.
3)	User selects the game they would like to remove and then hit the remove button
Termination Outcome:	The game is removed from the library, and the user is sent back to the main menu screen.

2.3. Interface Mockups



Name: Undertale

This is the description for popular Independent Video Game Undertale. Does this automatically wrap or

Last Played: 12/25/2016 Last Updated: 12/24/2016 Game Size: 1 GB

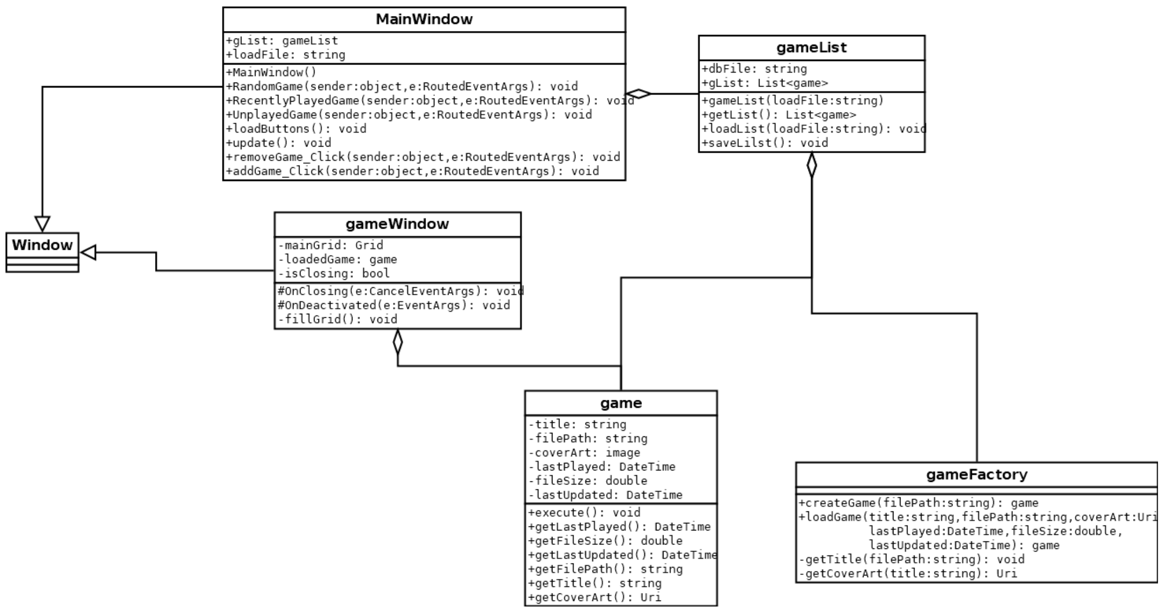
3. Project Timeline

Throughout the course of the project, we followed the Waterfall approach. First, we set requirements, in which we got together and created a proposal draft for our project that talks about background, impacts, challenges of our project,stretch goals, and a brief introduction of our project (September 13). Secondly, we started our design phase where we begin to layout our thoughts on design patterns and created a project structure and UML Outline (October 28). We spent the time until demo day working on our hard coded demo for class. Later after our Demo day, we were able to take our feedback from the audience to apply it towards updating our diagram structure to finish up our designs and start implementing what we had previously just hard-coded. There were various ups and downs in our plans, but by the end of the semester, we presented our completed product.

4. Project Structure

In this project, we wanted to make game management easier and more flexible for the user. In addition to that, we wanted to create a better experience, avoid wasting time looking through folders, and avoid wasting money buying a game they already own just because they weren't able to find it in their launcher. There is two windows: a Main for storing and accessing the games, and a secondary Game window to post any additional information present in the game and give the user the ability to play the game.

4.1. UML Outline


MainWindow
+gList: gameList
+loadFile: string
+MainWindow()
+RandomGame(sender:object,e:RoutedEventArgs): void
+RecentlyPlayedGame(sender:object,e:RoutedEventArgs): void
+UnplayedGame(sender:object,e:RoutedEventArgs): void
+loadButtons(): void
+update(): void
+removeGame_Click(sender:object,e:RoutedEventArgs): void
+addGame_Click(sender:object,e:RoutedEventArgs): void
gameList
+dbFile: string
+gList: List<game>
+gameList(loadFile:string)
+getList(): List<game>
+loadList(loadFile:string): void
+saveLilst(): void
Window
gameWindow
-mainGrid: Grid
-loadedGame: game
-isClosing: bool
#OnClosing(e:CancelEventArgs): void
#OnDeactivated(e:EventArgs): void
-fillGrid(): void
game
-title: string
-filePath: string
-coverArt: image
-lastPlayed: DateTime
-fileSize: double
-lastUpdated: DateTime
+execute(): void
+getLastPlayed(): DateTime
+getFileSize(): double
+getLastUpdated(): DateTime
+getFilePath(): string
+getTitle(): string
+getCoverArt(): Uri
gameFactory
+createGame(filePath:string): game
+loadGame(title:string,filePath:string,coverArt:Uri
lastPlayed:DateTime,fileSize:double,
lastUpdated:DateTime): game
-getTitle(filePath:string): void
-getCoverArt(title:string): Uri

4.2. Design Patterns Used

Factory pattern and Observer pattern.

5. Results

For our project, we have created a user-friendly game library manager that allows the user to organize and play their games. The user is able to add and delete games, view their most recently played game, a recommended game, and an unplayed game on the main interface. In a secondary window, which pops up once a game is selected, provides a brief description of that game, the last time it was played and updated, and the size of the game. We had intended to provide cover art of the games, however, we were unable to get this to functionally work.

5.1. Future Work

Now that everything is over, we're going to step away from the project. Steven might touch up some stuff out of spite, and see about implementing ideas mentioned in class, but otherwise we are content with letting the project end with the class.