KEYFRENZY

DESIGN MANUAL

Welcome to our game! Key Frenzy is a dynamic typing game designed using JavaFX, integrating elements from both gaming and educational software to enhance typing skills and reaction times.

The game's core revolves around a player typing words correctly to "destroy" on-screen ghosts, each labeled with words that appear randomly and increase in difficulty as the player advances through levels. The gameplay environment is set up using a GridPane within a VBox layout, where animated ghosts traverse the screen, challenging the player to type the associated words before the ghosts reach the center of the screen. Ghost movements and interactions are managed by an AnimationTimer and PathTransition, ensuring smooth and responsive gameplay. The game's architecture follows the MVC (Model-View-Controller) pattern, with the KeyFrenzyView class handling the UI components and user interactions, while game logic and state are managed in separate controller and model classes.

Additionally, a custom WordDictionary class loads words from a file into a map, categorized by word length, to dynamically adjust the game's difficulty based on the player's level. Player progress, including score and health, is continuously updated and displayed, with the game's state (paused, ongoing, or ended) controlled via UI elements like buttons. The game concludes either when the player runs out of health or completes all levels, with a transition to a game over screen managed through JavaFX's FXMLLoader.

1. User's story

- Lana Del Rey/38 years old/ Mediocre typer - *main targeted demographic*
  Singer and artist. Does not have much exposure to technology in her life prior but wanted to actively (use PicsArt to make her album cover), try to learn to adapt herself to modern technology, which requires typing on a computer. Love calming, accessible features and simple user interface, simple and explicit instruction and navigation.
  "You always keep on learning in your life"

- Katya Petrova/13 years old/ Language learner - *main targeted demographic*
  Middle school student. First language is Russian, only familiar with the Cyrillic alphabet. Trying to learn English and be familiar with English/Latin alphabet typing. Prefer simple and

visually explicit interface so that they would not be hindered from gameplay by language barrier.
"Я люблю изучать английский"

- John Hatlen/22 years old/ Killer - *accommodated demographic*
  Professional gamer/ streamer. Competitive and loves to assert dominance over other players, and wants to compete against other people all the time. Looking for skill-based matchmaking and efficient gameplay to result in the best outcome.
  "Conquest is life. Winning is prophecy"

- Madison Denys/ 39 years old/ Zen Gamer - *accommodated demographic*
  Having 3 children, was a social worker, and needed some means of peaceful recreation in her free time to relax and unwind, relieving stress. She loves games with calming visuals and theatrics, and repetitive tasks.
  "I play video games recreationally and I am not too enthusiastic"

- Phoebe Bates /19 years old/ Adrenaline Junkie - *accommodated demographic*
  College student. Casual game player for amusement. Love fast-paced gameplay, skill acquiring and thrilling visuals.
  "It is all fun and games"

2. Object Oriented Design

For KeyFrenzy's OOD, one could break down the system into classes that handle differentiated aspects of the game. Here is a detailed documentation of such:

CRC Cards

Dictionary (Interface?)	
Get a random word and return it(pulls from the csv file)	

These are the CRC cards that best represent the initial planning and design of the game

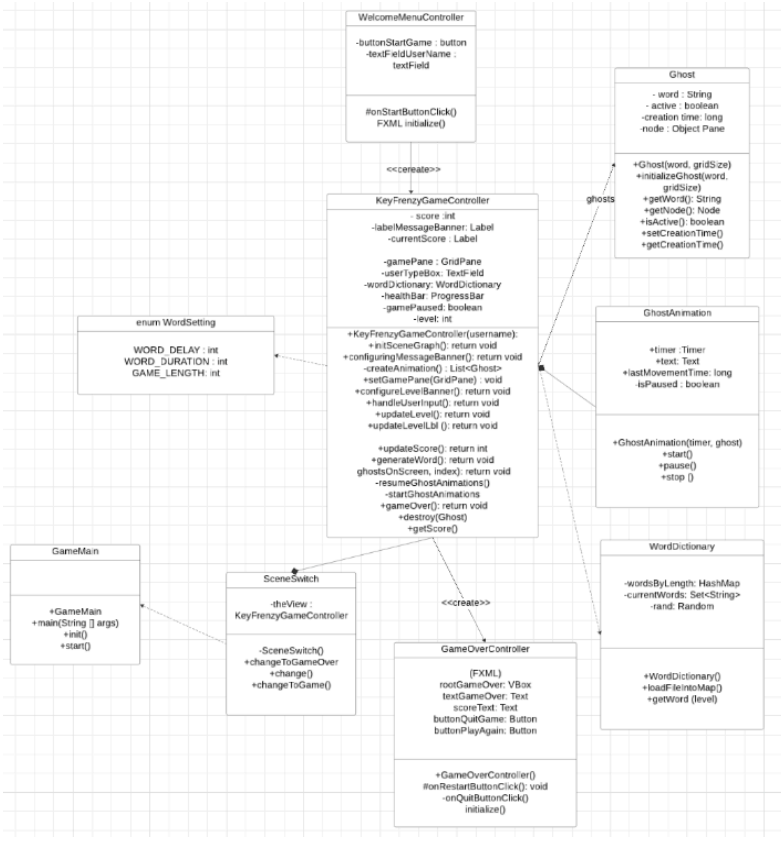
Words	
- Display the words (on top of the ghosts) - Ask the user for a guess and return it - Set the word to display on the main screen	

GameController	
- Get the words on ghosts to display - Start a new game by setting up the main character - Display the words on top of the zombies - Ask the user for a guess - Update the score of the main char (Increase the difficulty level [longer words/faster speed]) - Display the hearts remaining	- Dictionary - MainCharacter - Ghosts, Words - Words - MainCharacter

MainCharacter (User's input for their nickname)	
- Initialize the character - Display the character - Update the current life status (dead after getting attacked 3 times)	(most likely none?)

Ghosts	
- Initialize the 4 ghosts - Make the ghost disappear if the user enters the word correctly - Moving towards the target - Attack the target if distance is 0	- Words

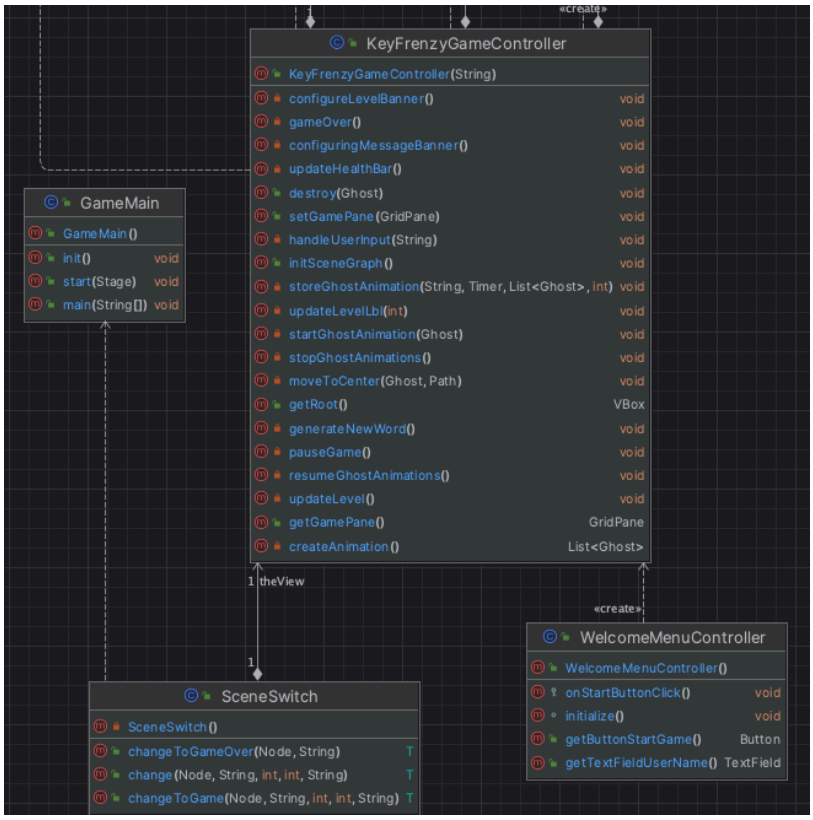
Planned Object Oriented Design for KeyFrenzy, adapted from LucidChart



GameController classes:

GameMain class:

Launching the game and necessary components, including FXMLLoader for the WelcomeMenu.fxml



KeyFrenzyController class:

Handles user interface components.

Utilizes JavaFX for UI layout and event handling.

Contains methods to update UI elements such as score, health, and level.

Displays the game screen, including the GridPane for ghost movement and word display.

Controls the game flow and logic.

Implements the MVC pattern's controller component.

Manages user input and triggers actions such as typing verification and ghost destruction.

Updates the game state based on player actions and events.

Represents the game's data and state.

Stores information such as player score, health, current level, and game progression.

Manages the game's difficulty progression and word generation.

Communicates with the WordDictionary class to retrieve words for ghosts.

SceneSwitch class:

Communicate between KeyFrenzyGameController, WelcomeMenuController and GameOverController classes to switch scenes as prompted.

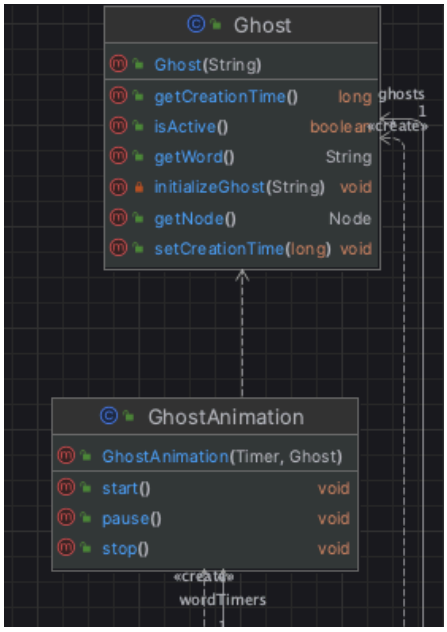
WelcomeMenuController class & GameOverController class:

Handles the loading of FXML files for UI layout. Facilitates transitions between different game screens, such as the main game screen and game over screen.

Ghost classes:

Ghost class:

Represents the ghost entity in the game and initializes all the visualization for the ghost.



GhostAnimation class:

Contains properties such as position, speed, and associated word labels. Handles movement animation using JavaFX's PathTransition.

Triggers actions upon reaching the center of the screen or being destroyed by the player. Controls the game's animation loop.

TypingMechanism classes:

WordDictionary class:

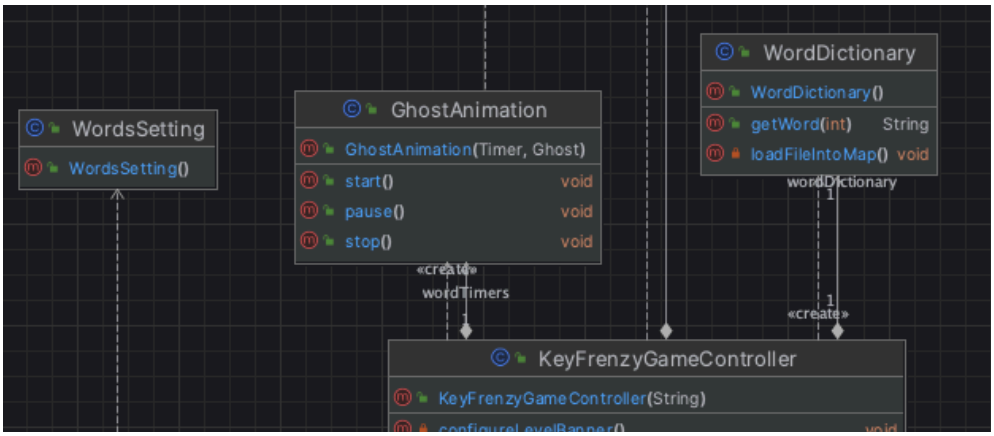
Loads words from a file into memory.

Categorizes words by length to facilitate dynamic difficulty adjustment.

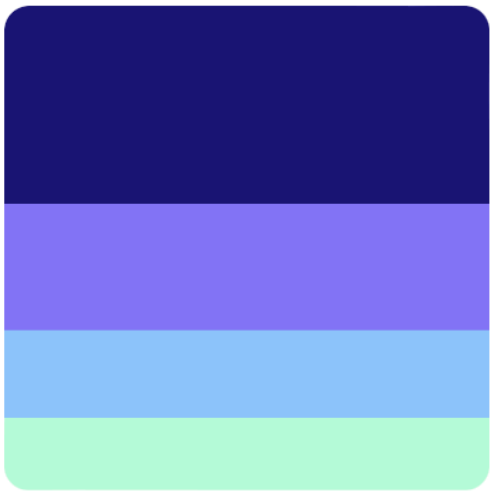
Provides methods to retrieve words for ghost labels based on the current game level.

WordsSetting class:

Instantiate word delay, duration and game length as a word setting. This class is particularly linked to balance out all the user personas, of which we have to find the equilibrium of challengingness and accessibility to the main demographic for the game, which was typer-learners.



Other design choices



High contrast color palette with cool colors to ensure accessibility for all demographics while retaining visual invigorating ness.

Futura

Futura is the main utility font. It's a serif typeface to ensure maximum readability and accessibility for all demographics.

VALORAX

Valorax was chosen as a title and embellishment font, due to its arcade-esque nature while still being a solid serif font for accessibility/readability.