

---

# Pocket Penguin Documentation

---

Version 1.0

July 22nd, 2024

Kayla Toliver



## Table of Contents

---

1.	<a href="#">Project Overview</a>	2
2.	<a href="#">Current Items</a>	3
3.	<a href="#">Technologies Used</a>	4
4.	<a href="#">User Guide</a>	5
5.	<a href="#">System Overview</a>	7
6.	<a href="#">Component Breakdown</a>	9
7.	<a href="#">Version History</a>	14

# 1. Project Overview

---

Pocket Penguin is a simple web-based game in which you interact with a small penguin inside an igloo. The goal is to keep the penguin's mood up. To do this, the user can buy items using the in-game currency, snow, and use them on the penguin to increase its mood.

## Mood Meter

Located in the top left panel. Consists of 3 hearts. The game starts with two hearts filled. Every 10 seconds, the mood meter decreases by 1.

## Snow

Located in the top right panel. Currency used in-game. Snow is awarded when the user interacts with (clicks) the penguin. The amount of snow given depends on the mood:

- ❖ Three hearts: +4 snow.
- ❖ Two hearts: +3 snow.
- ❖ One heart: +2 snow.
- ❖ Zero hearts: +1 snow.

## Items

Items come in three categories: food, toys, and decor. Each item costs a certain amount of snow and offers either a certain amount of mood points or time to which the mood meter decreases. Each item category has its own menu that holds its respective item (collectively called the store, located on the left-hand panel). Once bought, the item can be accessed in its inventory located on the right panel: eat, play, and furnish.




- ❖ **Food:** Can be bought any number of times. Provides a certain number of mood points which are automatically given to the penguin upon use. Can be accessed in the Eat menu.

- ❖ **Toys:** Functions similarly to food, except for minigames, which will not award the mood points unless completed. Can be accessed in the Play menu.
- ❖ **Decor:** Can only be bought once. Provides extra time to the mood meter. Once in its inventory, it can be added or removed from the penguin's igloo. The added time is only applied when the item is placed in the igloo. Can be accessed in the Furnish menu.


## 2. Current Items (v0.1)



---

### Food




Name	Image	Snow	Hearts
Shrimp	 A blue-bordered icon showing a shrimp. At the top left is '2*' and at the top right is '1♥'. Below the shrimp is the word 'SHRIMP'.	2	1
Fish	 A blue-bordered icon showing a fish. At the top left is '3*' and at the top right is '1♥'. Below the fish is the word 'FISH'.	3	1
Squid	 A blue-bordered icon showing a squid. At the top left is '4*' and at the top right is '2♥'. Below the squid is the word 'SQUID'.	4	2

### Toys

Name	Image	Snow	Hearts
Sled	 A blue-bordered icon showing a sled. At the top left is '10*' and at the top right is '2♥'. Below the sled is the word 'SLED'.	2	1

Skates		10	2
Tic-Tac-Toe		30	3 (if completed)

## Decor

Name	Image	Snow	Mood Meter Time
Couch		50	+5 sec
Lights		70	+10 sec
Fish Tank		100	+20 sec

## 3. Technologies Used

---

- ❖ **Front-end:** HTML, CSS
- ❖ **Back-end:** Javascript
- ❖ **Art and assets:** Procreate

## 4. User Guide

---



### Start Screen/Main Menu

The user can access any six of the menus as well as interact with the penguin, increasing snow.



### Food Menu

The user can exchange snow for food items.



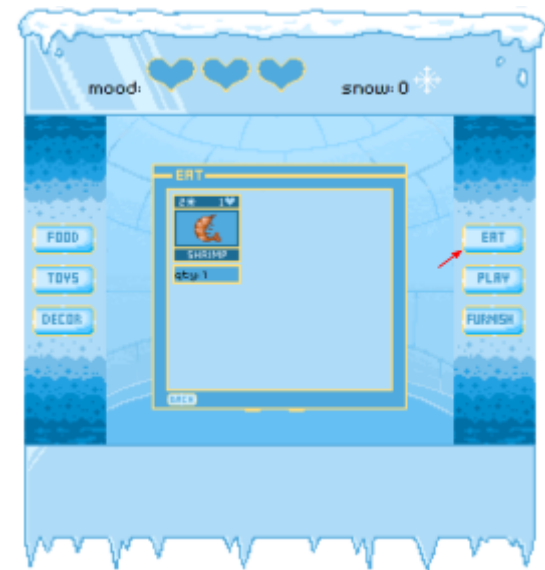
### Toys Menu

The user can exchange snow for toy items.



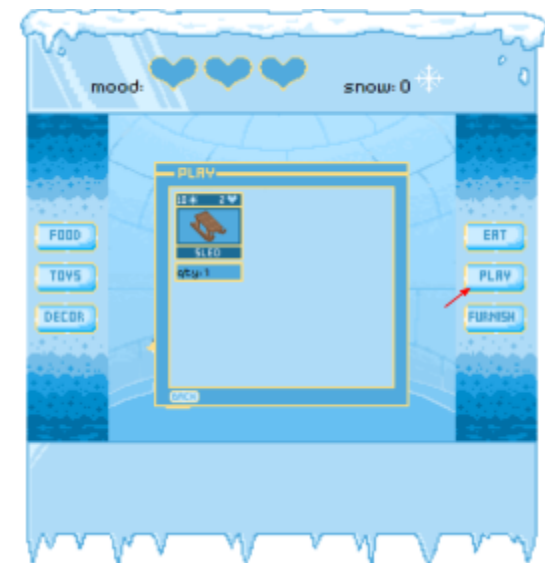
## Decor Menu

The user can exchange snow for decor items. Can only be purchased once.



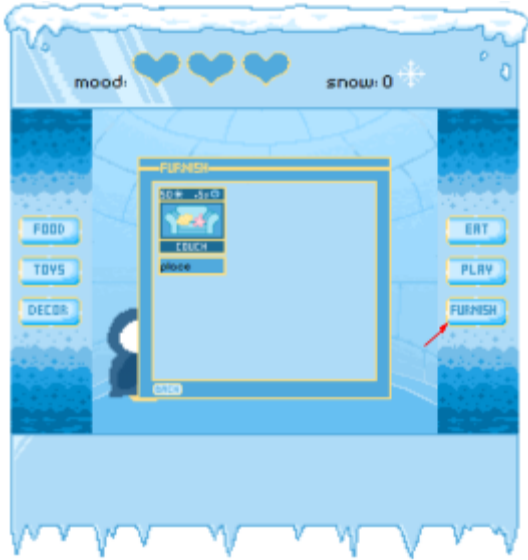
## Eat Menu

The user can use food items bought from the Food menu.



## Play Menu

The user can use toy items bought from the Toys menu.

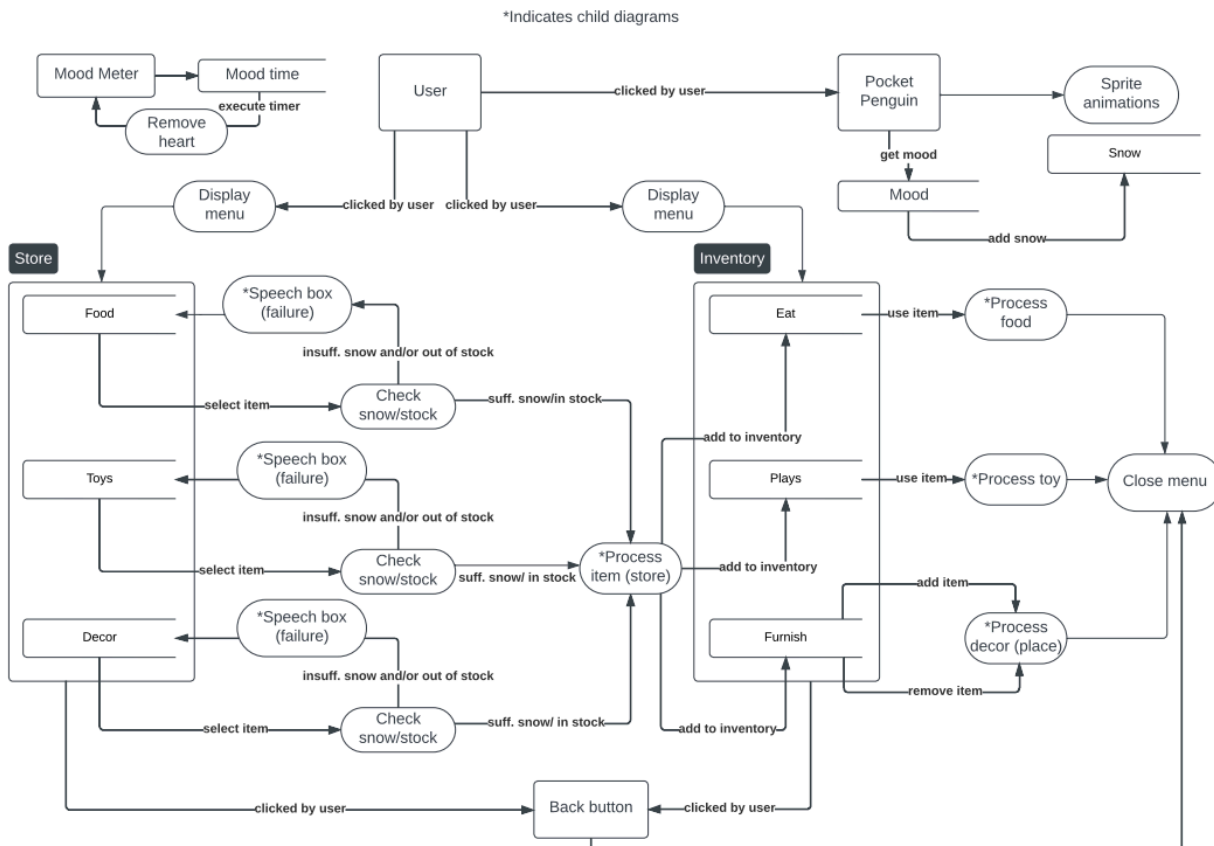


## Furnish Menu

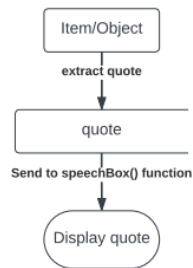
The user can add decor items to the igloo or return them to the Furnish menu. These items were bought from the Decor menu.

## 5. System Overview

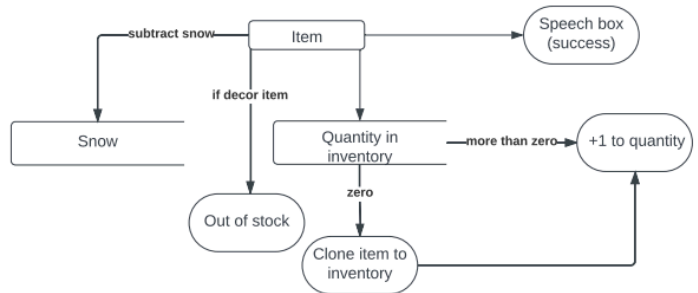
Pocket Penguin Data Flow Diagram



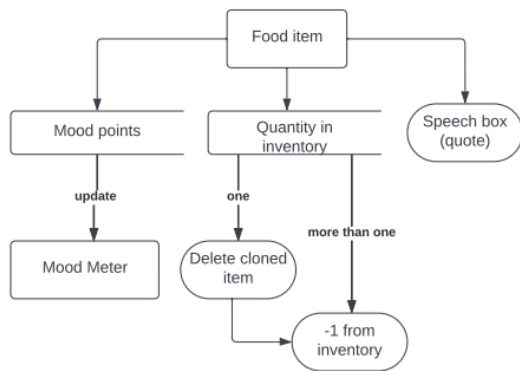
### Speech box



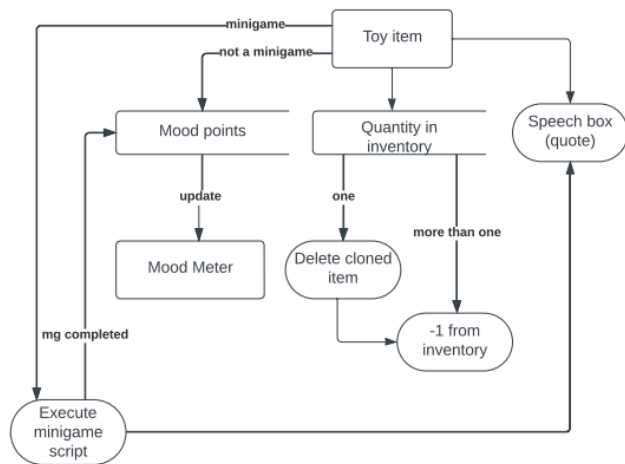
### Process item (store)



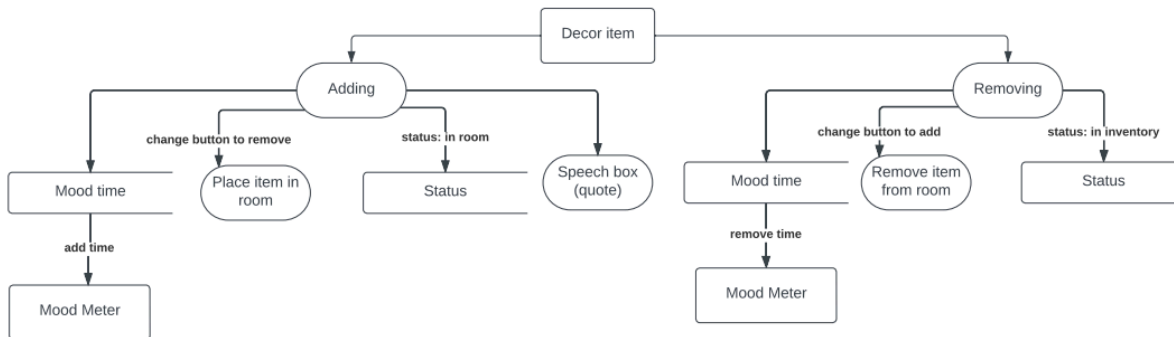
### Process food



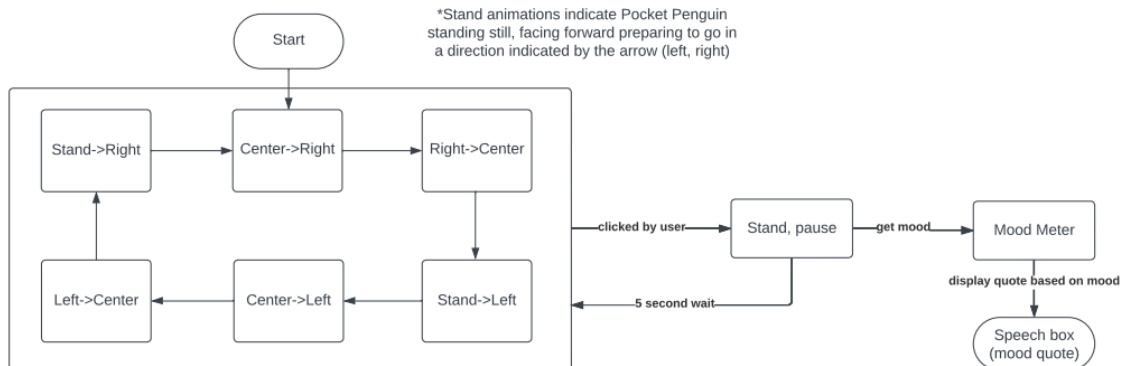
### Process toy



### Process decor



### Sprite Animations





## 6. Component Breakdown

---

### Files

- ❖ **index.html:** Contains the layout for the front-end. Each art asset containing a main game feature is in a div, including but not limited to the ice block containing the stats (snow and Mood Meter), the background (containing the penguin and furniture), the talk box, the store (food, toy, and decor buttons), and the inventory (eat, play, and furnish menus).
- ❖ **style.css:** Contains the art and styling for the front-end. Art assets are applied utilizing the “background: url()” rule. Main font used is Pixelify Sans.
- ❖ **main.js:** Contains the functionality for the front-end and the data for the back-end. Provides functions for all the main game features including the speech box, mood meter, hearts, items, store, inventory, penguin interactions, and penguin animations.
- ❖ **tic-tac-toe.js:** Contains the functionality for the tic-tac-toe minigame. Contains an easy mode where the AI selects spaces randomly, and a hard mode where the AI plays optimally (minimax algorithm).

### Objects (main.js)

- ❖ **Heart:** Determines the behavior of a heart in the mood meter. Contains functions that fill and empty the heart. In the context of this program, three heart objects were created and pushed into an array, “mood,” acting as the mood meter. The array is then sent as an argument to functions that increase or decrease the mood meter.
- ❖ **Item:** Determines the behavior and properties of an item. Because decor items contain a few more properties than food and toys, these properties are left blank when defining the latter. The properties in the Item class include the

name of the item, the price, the button (the DOM of the item id which includes the item's art asset), the item type, the number of hearts it provides, a custom quote when the item is used on the penguin, the quantity in the inventory, the status, whether it is in or out of stock, the image if it is used in the background, and the DOM element for the previous property. Each property has setter and getter functions. This class also includes a function that checks if the user has enough snow to purchase the item, returning a boolean. In the context of this program, nine Item objects were created: shrimp, fish, squid, sled, skates, tic-tac-toe, couch, lights, and tank. All of these objects were pushed into a storeItems array so that an event listener could be used for store/purchasing functionalities. These objects are also frequently utilized in the addToInventory and addDecor functions.

## Functions (main.js)

- ❖ **textBox():** Includes a parameter for the text to be displayed on the text box. Other than the starting message, each message is displayed for 5 seconds before the text box is cleared out. If another message is passed into the function while the current message is being displayed, any existing timeout is cleared so that the last passed message is allotted the full time for viewing.
- ❖ **moodDecreaser():** A helper function. Includes a parameter for the mood meter array containing the three Heart objects. A decrementing for loop is used to empty the last full Heart object.
- ❖ **startMoodDecreaser():** Prompts the moodDecreaser() function after a certain time interval. The initial time interval is 10 seconds, however this may change depending on the decor items placed. Any existing time intervals are cleared to avoid multiple hearts being decreased within the current interval.
- ❖ **moodIncreaser():** Includes parameters for the mood meter array and the number of hearts an item provides. This function goes through the mood meter with a for loop and fills in any number of hearts given by the hearts

parameter. It then calls the `startMoodDecreaser()` function to clear any existing time interval.

- ❖ **addToInventory():** Includes a parameter for an Item object. Primarily for food and toy items. This function clones the item given by the parameter and adds it to its respective inventory menu. Additionally, it adds a DOM object of the quantity of the item. If the item already exists in the inventory, the quantity is increased. Additionally, an event listener is applied to the cloned item so that when the penguin is clicked, the hearts are given through the `moodIncreaser()` function and the quote is passed to the `textBox()` function. If the item is a minigame (such as tic-tac-toe), a promise is initiated so that the hearts are not awarded until the game is completed.
  - During development, I had some trouble updating the quantity in the DOM. Because of this, I had to create a helper function that ensures the quantity is updating correctly in the DOM when switching between the store and the inventory.
  - \*In future updates, I think it would be better to copy and paste the items from the store to the inventory and use a tag for the quantity in the HTML rather than using DOM manipulation. It would probably be simpler and more straightforward than constantly cloning and deleting DOM objects in JS.
- ❖ **updateInventory():** Helper function that ensures every item's quantity is displayed correctly in the front-end when switching between the store and the inventory. It retrieves each item button from the DOM, then it retrieves the `p` tag that contains the quantity, then updates the quantity retrieved from the respective object.
- ❖ **addDecor():** Includes a parameter for an Item object. Primarily for decor. Functions similarly to `addToInventory` except instead of storing the quantity in a new DOM object, it stores the state (whether it is in the inventory or in the igloo). When the item button is clicked, the state is toggled.

- ❖ **checkSnow():** A function included in the Item class that checks if the user has enough snow to purchase the item, and updates the snow if they do.
- ❖ **getQuote():** Includes a parameter for an array of quotes. Selects and returns a random quote from the array.
- ❖ **penguinInteract():** Includes a parameter for the mood meter. Checks the mood meter and returns a quote by calling the getQuote() function and passes it as an argument to the textBox() function.
- ❖ **centertoRight():** Sprite of the penguin walking from the center to the right, has an event listener to call righttoCenter() upon completion.
- ❖ **righttoCenter():** Sprite of the penguin walking from the right to the center, has an event listener to call standtoLeft() upon completion.
- ❖ **standtoLeft():** Sprite of the penguin standing in the center, has an event listener to call centertoLeft() upon completion.
- ❖ **centertoLeft():** Sprite of the penguin walking from the center to the left, has an event listener to call lefttoCenter() upon completion.
- ❖ **lefttoCenter():** Sprite of the penguin walking from the left to the center, has an event listener to call standtoRight() upon completion.
- ❖ **standtoRight():** Sprite of the penguin standing in the center, has an event listener to call centertoRight() upon completion.
- ❖ **pauseAnimation():** When the penguin is clicked, the current sprite animation will be paused, stored in a variable, then will display a penguin standing sprite for three seconds. After the three seconds have elapsed, the current sprite animation will resume.

## Functions (main.js):

- ❖ **ticTacTextBox():** A `textBox()` function for the tic-tac-toe minigame.
- ❖ **clearBoard():** A function that clears each space on the game board. Used upon completion of a game.
- ❖ **endGame():** Includes a parameter for the winner of the game. Will pass a congratulatory dialogue to the winner as an argument for the `ticTacTextBox()` function.
- ❖ **isFull():** Checks each space of the game board grid to check if it is full or not. Also sets the `ticTacToeFinished` boolean so the program knows whether to award the hearts to the user.
- ❖ **checkWin():** Includes a parameter for the current spot. Checks every possible winning combination (row, column, diagonals) in the game board grid with the current chosen spot.
- ❖ **markSpace():** Includes parameters for the mark (X or O) and the current space. Marks the current chosen space in the game board with the player or opponent's mark.
- ❖ **checkWinTestGrid():** Includes parameters for the grid and the player's mark. Works similarly to the `checkWin()` function, instead of checking the actual game board itself, it checks the grid that holds the game information. Primarily used for checking a win for the minimax algorithm.
- ❖ **evaluate():** Helper function for checking the winner in for the minimax algorithm.
- ❖ **isTestGridFull():** Includes parameters for the grid. Works similarly to the `isFull()` function, instead of checking the actual game board itself, it checks the grid

that holds the game information. Primarily used for checking a win for the minimax algorithm.

- ❖ **minimax():** An algorithm that assumes the AI plays optimally during a game of tic-tac-toe. Used for hard mode. It checks each empty space. For each space it marks the opponent, then recursively checks every possible move for the remainder of the game, storing the best move in a variable.
- ❖ **hardMode():** Helper function that utilizes the minimax() function. Finds the best possible move for the opponent, performing the minimax algorithm for each free space.
- ❖ **easyMode():** AI/opponent chooses a random spot on the game board.
- ❖ **startGame():** Sets up the game board and allows the user to make a move. It then calls either easyMode() or hardMode() (depending on what the user chooses) so the AI/opponent can make its move.
- ❖ **ticTacToe():** Starter function that provides a callback to the promise made in main, allows the user to select a difficulty then calls startGame() immediately after.

## 7. Version History (v0.1)

---

### 7/14/2024 - v0.1

- ❖ Food items: Shrimp, fish, squid
- ❖ Toy items: Sled, skates, tic-tac-toe (minigame)
- ❖ Decor items: Couch, lights, fish tank
- ❖ Current bugs:

- Can still make moves on tic-tac-toe easy mode game board once the game is completed.
- Mood meter time addition not tested, does not automatically update in some cases.

❖ To do:

- Redesign logo.
- Redesign quantity UI.