battleMonsters Documentation

Luke Renchik
Built on My Desk

February 11, 2024

Abstract

This document serves as the official documentation for `battleMonsters`, a command-line based game inspired by the popular Pokémon series. Developed in C, `battleMonsters` offers players the experience of navigating an overworld and engaging in battles with various monsters. This documentation provides a comprehensive guide on the installation, usage, architecture, and functionalities of `battleMonsters`, aimed at both users and developers.

Contents

1 Introduction

`battleMonsters` is a text-based adventure and strategy game that allows players to explore a virtual world, capture, and battle with monsters in a manner reminiscent of Pokémon. Written in the C programming language, it emphasizes simplicity, portability, and accessibility, making it easy to run on various operating systems with minimal dependencies.

The game is designed to be engaging and challenging, offering a blend of exploration, strategy, and combat. Players navigate through the game's overworld, encountering wild monsters and other trainers, with the goal of becoming the ultimate monster trainer.

This documentation is structured to provide a thorough understanding of `battleMonsters`' features, including detailed descriptions of its programming interfaces, usage examples, and guides on how to extend and contribute to the project. Whether you are a player seeking to learn more about the game mechanics or a developer interested in the game's codebase, this documentation aims to provide the necessary information to enhance your experience with `battleMonsters`.

2 System Requirements and Installation

This section will detail the system requirements and step-by-step installation instructions for `battleMonsters`.

3 Usage

This section will explain how to start and play the game, including an overview of the command-line arguments and configuration options.

4 Game Architecture and API Documentation

This section delves into the core functionalities provided by `battleMonsters`, focusing on the primary functions within the game's codebase. Detailed below are the prototypes, usage examples, and operational details of pivotal functions.

4.1 Function: `addAbility`

4.1.1 Description

Adds an `Ability` to a `battleMonster`. Each `battleMonster` is limited to a maximum of `MAX_ABILITIES` abilities. If this maximum is reached, the function prompts the user to replace an existing ability or to cancel the addition.

4.1.2 Parameters

- `battleMonster *monster` - A pointer to the `battleMonster` which will receive the new ability.
- `Ability *ability` - A pointer to the `Ability` to be added.

4.1.3 Usage

```
addAbility(&monster, &ability);
```

4.1.4 Important Notes

The function involves user interaction for selecting an ability to replace if the maximum number of abilities is already reached. The selection is made via standard input.

4.2 Function: `battleMonsterInitialization`

4.2.1 Description

Initializes a `battleMonster` with specified attributes including its name, starting health, level, type, and speed. It also prepares the monster to store abilities, initializing the count to zero and allocating memory for them.

4.2.2 Parameters

- `battleMonster *name` - Pointer to the `battleMonster` to initialize.
- `int startingHealth` - The initial health of the `battleMonster`.
- `int level` - The starting level of the `battleMonster`.
- `char type[]` - The type of the `battleMonster` (e.g., "Water", "Fire").
- `char monsterName[]` - The name of the `battleMonster`.
- `int bmSpeed` - The speed attribute of the `battleMonster`.

4.2.3 Usage

```
battleMonsterInitialization(&monster, 100, 1,
    ↪  "Fire", "Flaremon", 10);
```

4.2.4 Important Notes

This function sets the `battleMonster`'s experience to 0 and current health equal to the starting health. Memory allocation for storing up to `MAX_ABILITIES` is performed, with error handling for allocation failure.

5 Battle Sequence API Documentation

This section outlines the functionalities related to the battle sequence within `battleMonsters`, detailing the key functions that manage battles between monsters, damage calculation, and user interaction during battles.

5.1 Function: `endBattle`

5.1.1 Description

Terminates a battle between the player's monster and the opponent's monster, determining the outcome based on the current health of each monster. It also deactivates the battle state.

5.1.2 Parameters

- `battleMonster *userMonster` - The player's monster.
- `battleMonster *oppMonster` - The opponent's monster.
- `Player *user` - The player.

5.1.3 Notes

If either monster's health drops to 0, the battle ends with a message indicating the outcome. The function also triggers any post-battle processes, such as awarding loot to the player.

5.2 Function: `clearInputBuffer`

5.2.1 Description

Clears the input buffer to ensure that unintended input does not affect subsequent reads from `stdin`.

5.2.2 Parameters

None.

5.3 Function: `runFromBattle`

5.3.1 Description

Allows the player to attempt to flee from a battle, based on the type of battle. If successful, it ends the battle; otherwise, it prevents the escape.

5.3.2 Parameters

- `char bType` - The type of battle, indicating whether escape is possible.
- `battleMonster *userMonster` - The player's monster.
- `battleMonster *oppMonster` - The opponent's monster.
- `Player *user` - The player.

5.3.3 Notes

Escape is only possible in certain types of battles, denoted by `bType`.

5.4 Function: `calculateDamage`

5.4.1 Description

Calculates and applies damage from an attacking monster to a defending monster based on the selected ability and random factors to simulate battle variability.

5.4.2 Parameters

- `battleMonster *attacker` - The attacking monster.
- `battleMonster *defender` - The defending monster.
- `Ability *selectedAbility` - The ability used in the attack.

5.4.3 Notes

Damage calculation includes randomness to reflect the unpredictable nature of battles. The function also displays the outcome of the attack.

5.5 Function: `selectAbility`

5.5.1 Description

Prompts the user to select an ability for their monster to use during a battle. Validates the user's choice and executes the selected ability if valid.

5.5.2 Parameters

- `battleMonster *userMonster` - The player's monster.
- `battleMonster *oppMonster` - The opponent's monster.

5.5.3 Notes

The function ensures that the chosen ability is within the valid range of the monster's abilities and handles user input accordingly.

5.6 Function: `opponentAbilitySelect`

5.6.1 Description

Automatically selects an ability for the opponent's monster based on a random choice, simulating an AI opponent's decision-making in battle.

5.6.2 Parameters

- `battleMonster *oppMonster` - The opponent's monster.
- `battleMonster *playerMonster` - The player's monster.

5.6.3 Notes

The selection is pseudo-random, aiming to add unpredictability to the opponent's actions during a battle.

5.7 Function: `printBattleState`

5.7.1 Description

Prints the current state of the battle, including each monster's name and their respective current and starting health.

5.7.2 Parameters

- `battleMonster *userMonster` - The player's monster.
- `battleMonster *oppMonster` - The opponent's monster.

5.8 Function: `startBattle`

5.8.1 Description

Initiates a battle sequence, setting up initial conditions such as turn order based on monster speed and entering the battle loop until the battle concludes.

5.8.2 Parameters

- `ActionContext* context` - Context containing both player and opponent monsters, and additional battle-related information.
- `Player* player` - The player.

5.8.3 Notes

This function orchestrates the overall battle sequence, managing turn order, player input, and battle progression until an end condition is met.

6 Items API Documentation

This section covers the functionalities related to the usage of items within `battleMonsters`. Items can provide health recovery or facilitate escape during battles, enhancing the strategic options available to players.

6.1 Function: `useHealthPotion`

6.1.1 Description

Restores a specified amount of health to a `battleMonster`. If the healing exceeds the monster's maximum health, its current health is capped at its starting health value.

6.1.2 Parameters

- `battleMonster *monster` - The monster receiving the health restoration.

6.1.3 Usage

```
useHealthPotion(&monster);
```

6.1.4 Notes

Heals the monster for 25 hit points. Ensures the monster's health does not exceed its starting health.

6.2 Function: `useSuperHealthPotion`

6.2.1 Description

Significantly restores health to a `battleMonster`, following the same logic as `useHealthPotion` but with a greater amount of health restored.

6.2.2 Parameters

- `battleMonster *monster` - The monster that is healed.

6.2.3 Usage

```
useSuperHealthPotion(&monster);
```

6.2.4 Notes

Heals the monster for 50 hit points. This function also ensures that the monster's health does not surpass its starting health.

6.3 Function: `useSmokeBomb`

6.3.1 Description

Provides the player with an option to escape from a battle, simulating the use of a smoke bomb to distract the opponent. The exact escape mechanism is subject to further implementation.

6.3.2 Parameters

- `battleMonster *monster` - The monster attempting to use the smoke bomb to escape.

6.3.3 Usage

```
useSmokeBomb(&monster);
```

6.3.4 Notes

Currently, this function primarily serves as a placeholder for escape functionality, indicating the intent to allow players to flee from engagements under certain conditions. The detailed escape logic needs to be implemented.

7 Overworld API Documentation

This section outlines the functionalities related to the overworld navigation and interactions within `battleMonsters`. It includes the management of map entities, player movement, and interactions with various objects and characters in the game world.

7.1 Function: `clearConsole`

7.1.1 Description

Clears the console to refresh the game's display, ensuring that the player sees an updated view of the overworld or battle screen.

7.1.2 Parameters

None.

7.1.3 Usage

```
clearConsole();
```

7.1.4 Notes

This function is platform-dependent and may need adjustments for compatibility with different operating systems.

7.2 Function: `mapPersonInitializationWithAction`

7.2.1 Description

Initializes a `mapPerson` entity on the game map with specified characteristics and an equipped battle monster. This entity can represent players, NPCs, or enemies.

7.2.2 Parameters

- `mapPerson *name` - A pointer to the `mapPerson` being initialized.
- `int xPos` and `int yPos` - The x and y coordinates for the entity's position on the map.
- `char text[]` - Text associated with the entity, typically used for interactions.
- `int ID` - A unique identifier for the entity.
- `battleMonster* equippedBM` - A pointer to a `battleMonster` that the entity has equipped.

7.2.3 Usage

```
mapPersonInitializationWithAction(&entity,
    ↪ 10, 20, "NPC Text", 1, &monster);
```

7.2.4 Notes

Entities equipped with battle monsters can engage in battles. The function places the entity on the global map at the specified coordinates.

7.3 Function: `addActionToMapPerson`

7.3.1 Description

Assigns an action function to a `mapPerson`, enabling the entity to perform specific actions when interacted with.

7.3.2 Parameters

- `mapPerson *name` - The `mapPerson` to which the action is being added.
- `ActionFunctionWithContext action` - A function pointer representing the action to be performed.
- `ActionContext* context` - A pointer to an `ActionContext` structure containing any additional information needed by the action.

7.3.3 Usage

```
addActionToMapPerson(&entity, actionFunction,
    ↪  &context);
```

7.3.4 Notes

This function allows for dynamic interactions within the game world, enabling entities to have custom behaviors.

7.4 Function: `clearFromMap`

7.4.1 Description

Removes a `mapPerson` from the map, effectively clearing its current position and making the cell available for other entities or leaving it empty.

7.4.2 Parameters

- `mapPerson *name` - The `mapPerson` entity to be removed from the map.
- `mapPerson* grid[MAX_X][MAX_Y]` - The grid representing the game map from which the entity is removed.

7.4.3 Usage

```
clearFromMap(&entity, map);
```

7.4.4 Notes

This function is crucial for updating the map's state, especially during entity movements or when entities are removed from the game (e.g., after a battle).

7.5 Function: `mapPersonMovement`

7.5.1 Description

Updates the position of a `mapPerson` on the map based on the specified direction. It checks for boundary conditions and collisions to ensure valid movement.

7.5.2 Parameters

- `mapPerson *person` - The `mapPerson` entity to move.
- `mapPerson* grid[MAX_X][MAX_Y]` - The game map grid.
- `char direction` - The direction of movement ('U' for up, 'D' for down, 'L' for left, 'R' for right).

7.5.3 Usage

```
mapPersonMovement(&player.location, map, 'U')
    ↪ ;
```

7.5.4 Notes

Ensures that movements are within map boundaries and prevents entities from moving into spaces occupied by other entities.

7.6 Function: `placeOnMap`

7.6.1 Description

Places a `mapPerson` entity on the map at its current coordinates, updating the map grid to reflect its new position.

7.6.2 Parameters

- `mapPerson *name` - The `mapPerson` entity to place on the map.
- `mapPerson* grid[MAX_X][MAX_Y]` - The grid representing the game map.

7.6.3 Usage

```
placeOnMap(&entity, map);
```

7.6.4 Notes

This function is used after moving an entity or initializing it to ensure the map accurately represents all entity positions.

7.7 Function: `displayMapFull`

7.7.1 Description

Renders the entire game map to the console, displaying entities at their respective positions.

7.7.2 Parameters

- `mapPerson* grid[MAX_X][MAX_Y]` - The grid representing the game map.

7.7.3 Usage

```
displayMapFull(map);
```

7.7.4 Notes

Useful for debugging or providing an overview of the entire map's state. Entities are marked with 'X', and empty spaces with '.'.

7.8 Function: `displayMapAroundPlayer`

7.8.1 Description

Displays a portion of the map centered around the player's position, providing a localized view of the immediate surroundings.

7.8.2 Parameters

- `mapPerson* grid[MAX_X][MAX_Y]` - The grid representing the game map.
- `int playerX` and `int playerY` - The x and y coordinates of the player's position.

7.8.3 Usage

```
displayMapAroundPlayer(map, player.location.
    ↪ xPosition, player.location.yPosition);
```

7.8.4 Notes

This function is essential for creating a dynamic and engaging player experience by focusing on the player's immediate environment.

7.9 Function: `isActionAdjacent`

7.9.1 Description

Checks if there is an actionable `mapPerson` entity adjacent to the specified `mapPerson`, enabling interactions such as battles or dialogues.

7.9.2 Parameters

- `mapPerson* grid[MAX_X][MAX_Y]` - The game map grid.
- `mapPerson *person` - The `mapPerson` around which to check for adjacent actions.

7.9.3 Usage

```
mapPerson* adjacentEntity = isActionAdjacent(
    ↪ map, &player.location);
```

7.9.4 Notes

This function facilitates interactions within the game world, checking for entities within one cell of the specified location.

8 Player API Documentation

This section covers the functionalities related to the player within `battleMonsters`, including inventory management, player initialization, and battle monster assignment.

8.1 Structures and Types

8.1.1 `Player`

Represents the player in the game, including location, inventory, equipped battle monster, and other player-specific data.

8.1.2 `ActionContext`

Holds contextual information for actions that can be performed by or on the player, such as battle context.

8.1.3 `InventorySlot`

Defines a slot in the player's inventory, holding an item and its quantity.

8.1.4 `mapPerson`

Describes entities on the map, including the player, NPCs, and monsters, with properties for position, interaction text, and actions.

8.2 Function: `addItemToInventory`

8.2.1 Description

Adds an item to the player's inventory. If the item already exists, increases the quantity; otherwise, adds a new entry.

8.2.2 Parameters

- `Player *player` - The player whose inventory is being modified.
- `Item newItem` - The item to add to the inventory.
- `int quantity` - The quantity of the item to add.

8.2.3 Usage

```
addItemToInventory(&player, newItem, quantity
    ↪ );
```

8.3 Function: `printInventory`

8.3.1 Description

Displays the player's inventory, listing each item with its description and quantity.

8.3.2 Parameters

- `const Player *player` - The player whose inventory is to be displayed.

8.3.3 Usage

```
printInventory(&player);
```

8.4 Function: `selectInventoryItem`

8.4.1 Description

Allows the player to select an item from their inventory for use, such as healing potions or other usable items.

8.4.2 Parameters

- `Player *player` - The player selecting an item.
- `battleMonster *userMonster` - The battle monster that may benefit from the item use.

8.4.3 Usage

```
selectInventoryItem(&player, &userMonster);
```

8.5 Function: `initPlayer`

8.5.1 Description

Initializes player data, including inventory size, money, and starting location on the map.

8.5.2 Parameters

- `Player *player` - The player to initialize.
- `int invSize` - Initial inventory size.
- `int initMoney` - Initial amount of money.
- `int x`, `int y` - Starting coordinates on the map.

8.5.3 Usage

```
initPlayer(&player, 0, 100, 5, 5);
```

8.6 Function: `assignBattleMonsterToPlayer`

8.6.1 Description

Assigns a battle monster to the player, setting it as the currently equipped monster for battles.

8.6.2 Parameters

- `Player* player` - The player to assign the monster to.
- `battleMonster* battleMonster` - The monster to be assigned to the player.

8.6.3 Usage

```
assignBattleMonsterToPlayer(&player, &
    ↪ battleMonster);
```

8.7 Function: `getPlayerBattleMonster`

8.7.1 Description

Retrieves the battle monster currently equipped by the player.

8.7.2 Parameters

- `Player* player` - The player whose battle monster is being retrieved.

8.7.3 Usage

```
battleMonster* monster =
    ↪ getPlayerBattleMonster(&player);
```

8.8 Function: `assignActionContext`

8.8.1 Description

Assigns context for actions that can be performed by the player, such as battling with another monster.

8.8.2 Parameters

- `ActionContext* ac` - The action context to be filled.
- `char param` - A parameter characterizing the action.
- `battleMonster* playerBM` - The player's battle monster.
- `battleMonster* oppBM` - The opponent's battle monster.
- `Player* player` - The player.

8.8.3 Usage

```
assignActionContext(&ac, 'p', &playerBM, &
    ↪ oppBM, &player);
```

8.9 Function: `getLoot`

8.9.1 Description

Awards loot to the player after defeating an opponent monster, typically in the form of money.

8.9.2 Parameters

- `battleMonster *oppMonster` - The defeated opponent monster.
- `Player *user` - The player receiving the loot.

8.9.3 Usage

```
getLoot(&oppMonster, &player);
```

This comprehensive documentation provides detailed insights into the player component of `battleMonsters`, aiding in understanding and further development of the game's functionalities related to player interaction, inventory management, and player-monster dynamics.

9 Configuration and Customization

This section will outline how players and developers can configure or customize the game.

10 Troubleshooting and FAQs

This section will address common issues and questions regarding `battleMonsters`*.*

11 Contributing

This section will provide guidelines for those interested in contributing to the development of `battleMonsters`*.*