Programming Fundamentals

Submission

```
5206267 Luo, Zheng                              3707/3 AEROAH

Submissions:-

S 0     Wed Jul 22 22:32:58 2020        5206267 thu18b ass2_castle_defense -17:-19
S 1     Thu Jul 23 11:31:18 2020        5206267 thu18b ass2_castle_defense -17:-6
S 2     Thu Jul 23 18:38:41 2020        5206267 thu18b ass2_castle_defense -16:-23
S 0     Thu Jul 23 20:32:05 2020        5206267 thu18b ass2_castle_defense -16:-21
S 1     Thu Jul 23 23:41:51 2020        5206267 thu18b ass2_castle_defense -16:-18
S 2     Fri Jul 24 17:11:12 2020        5206267 thu18b ass2_castle_defense -16:0
S 0     Sat Jul 25 23:52:35 2020        5206267 thu18b ass2_castle_defense -14:-18
S 1     Sun Jul 26 13:14:30 2020        5206267 thu18b ass2_castle_defense -14:-4
S 2     Sun Jul 26 23:10:04 2020        5206267 thu18b ass2_castle_defense -13:-18
S 0     Mon Jul 27 01:11:10 2020        5206267 thu18b ass2_castle_defense -13:-16
S 1     Mon Jul 27 14:39:42 2020        5206267 thu18b ass2_castle_defense -13:-3
S 2     Mon Jul 27 14:58:14 2020        5206267 thu18b ass2_castle_defense -13:-3
S 0     Mon Jul 27 15:00:55 2020        5206267 thu18b ass2_castle_defense -13:-2

Mon Aug 17 10:05:57 2020                ## lyre12.orchestra.cse.unsw.EDU.AU ##
```

Listing

realm.c

```c
// Assignment 2 20T2 COMP1511: Castle Defense
//
// This program was written by Zheng Luo (z5206267@ad.unsw.edu.au)
// on July/2020
//
// Version 1.0.0 (2020-07-20): Assignment released.
                   ^
                   + ====================================== +
                   + Give a rundown of your approaches here +
                   + ====================================== +

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "realm.h"
///////////////////////////////////////////////////////////////////////

// `struct realm` represents a realm, which represents the state of the
// entire program. It is mainly used to point to a linked list of
// locations, though you may want to add other fields to it.
struct realm {
    struct location *castle;
    struct location *lair;
};

// `struct location` represents a location, which could be a land,
// a tower, or a castle. Each of those places has a different type
// of information that this struct will need to store.
struct location {
    char name[MAX_NAME_LENGTH];
    int power;
    int uses;
    int defense;
    struct enemy *enemies;
    struct location *next;
    Effect effect;
};

// `struct enemy` represents an enemy, which will move through the
// realm (to each location). Towers can deal damage to it, and
// it can deal damage to the Castle.
struct enemy {
    char name[MAX_NAME_LENGTH];
    int hp;
    int hp_Given;
          ^
          + ============================================== +
          + Make sure you stick to camelCase or snake_case +
          + ============================================== +
    struct enemy *next;
};


///////////////////////////////////////////////////////////////////////

// Function prototypes for helper functions
static Location new_location(char *name);
static void print_tower(char *name, int power, int uses, Effect effect);
static void print_land(char *name);
static void print_castle(char *name, int defense);
static void print_enemy(char *name, int cur_hp, int max_hp);
// Self-created functions below,
// its detailed introductions are located in Self-Created functions
// after stage 5 function.
static void remove_enemy(struct location *current, struct enemy *currentEnemy);
static int search_function_tower (struct location *current, char *search_term);
static int search_function_enemy (struct enemy *current, char *search_term);
static void sorting(struct location *current);

///////////////////////////////////////////////////////////////////////
```

```


// Create a new realm, and return a pointer to it.
// You may need to change this function in later stages.
Realm new_realm(void) {
    struct realm *realm = malloc(sizeof(struct realm));
    realm->castle = new_location("Castle");
    realm->lair = new_location("Lair");
    // Castle's defense health.
    realm->castle->defense = STARTING_CASTLE_HP;
    // Let lair pointing to NULL.
    realm->lair->next = NULL;
    // Let the enemy in castle and lair pointing to NULL.
             ^
             + ======================== +
             + Dont comment the obvious +
             + ======================== +
    realm->castle->enemies = NULL;
    realm->lair->enemies = NULL;

    if (realm->castle != NULL && realm->lair != NULL) {
        realm->castle->next = realm->lair;
    }

    return realm;
}


// Return a new location created with malloc.
static Location new_location(char *name) {
    // Allocate memory for new locations.
    struct location *place = malloc(sizeof(struct location));
    // Initiate the NULL pointer for place->enemy.
    place->enemies = NULL;
    // Initiate the variable for place->power and place->uses.
    place->power = 0;
    place->uses = 0;
    place->effect = EFFECT_NONE;
    // Assign the name into location.
    strcpy(place->name, name);
    // Return pointer for new location.
    return place;
}

////////////////////////////////////////////////////////////////////////
//                          Stage 1 Functions                         //
////////////////////////////////////////////////////////////////////////

// Add a new location to the realm, and return the total number of
// locations in the realm.
int add_location(Realm realm, char *name) {
    // Assign the new position as newLand.
    struct location *newLand = new_location(name);
    // Assign the corresponded name into newLand.
    strcpy(newLand->name, name);
    // Assuming 4 buildings.
    int counter = 4;
    // If no building in land yet, which means
    // castle is connecting to the lair.
    if (realm->castle->next == realm->lair) {
        // castle(head)->newLand->Lair->NULL
        newLand->next = realm->lair;
        realm->castle->next = newLand;
        counter--;
               ^
               + =============================== +
               + Why assume 4 and then subtract? +
               + =============================== +
    } else { // Land(s) exist between castle and
               ^
               + ================================================================= +
                + INLINE COMMENTS: it's usually better for comments to go on the    +
                + line above                                                         +
                + ================================================================= +
        // (head)->original    Lair->NULL
        //             |          |
        //              ->newLand->
        // Loop thro until the insert position is reached.

        // realm->land is same as head.
        // Here current should point to the first newLand, not NULL.
        struct location *current = realm->castle->next;
        while (current->next != realm->lair) {
            counter++;
            current = current->next;
        }
        newLand->next = current->next;
        current->next = newLand;

    }

    return counter;
}

// Print out the realm by looping through everything in the realm,
// including castle/land/tower/lair and enemies under each location.
// The print_realm function take realm as input,
// no output due to its void type function.
void print_realm(Realm realm) {
    // Pointer "current" start from first land.
    struct location *current = realm->castle;
    // Print out castle, lands and lair.
    // Loop thro until current reached NULL terminator at the end.
    while (current != NULL) {
        // Land will be printed if there is no power or uses.
        if (strcmp(current->name, "Castle") == 0) {
            // Print out castle.
            print_castle(realm->castle->name, realm->castle->defense);
        } else if (current->power == 0 || current->uses == 0) {
            print_land(current->name);
        }
        else {
            print_tower(current->name, current->power, current->uses,
            current->effect);
        }

        // If there's enemies under the location, proceed.
        if (current->enemies != NULL) {
            // Create a new pointer "currentEnemy" at current enemies,
            // in order to loop thro enemies under each location.
            struct enemy *currentEnemy = current->enemies;
            // Loop thro each enemy under the location
            while (currentEnemy != NULL) {
                // Print out enermies.
                print_enemy(currentEnemy->name,
                currentEnemy->hp, currentEnemy->hp_Given);
                // Move to the next enemy under the same location.
                currentEnemy = currentEnemy->next;
            }


        }
        current = current->next;
    }


}

////////////////////////////////////////////////////////////////////////
//                           Stage 2 Functions                          //
////////////////////////////////////////////////////////////////////////

// Add an enemy to the realm by looping through the list, and insert it
// at the end of the list.
// The function take realm, location name, the name of enemy, its hp as input.
// The function will return SUCCESS(0) if new enemy if successfully created.
int new_enemy(Realm realm, char *location_name, char *name, int hp) {
    // Ensure that the stats you have been given for HP are not below 1.
    // If they are, return ERROR_INVALID_STAT.
    if (hp < 1) {
        return ERROR_INVALID_STAT;
    }
    // Find the Location called location_name.
    // If one does not exist, return ERROR_NO_LOCATION
    // Create a pointer current, used to loop thro the list,
    // in order to match the same name.
    struct location *current = realm->castle;

    while (current != NULL) {
        if (strcmp(current->name,location_name) == 0) {
                                 ^
                                 + ============================================ +
                                 + Could this go in a function find_location()? +
                                 + ============================================ +
            // Place a new enemy, with the given stats,
            // directly after the last enemy at the location you found above.
            struct enemy *newEnemy = malloc(sizeof(struct enemy));
            // Create a pointer "currentEnemy",
            // in order to insert the new enemy.
            struct enemy *currentEnemy = NULL;
            // Assign the name and hp to the enemy.
            strcpy(newEnemy->name, name);
            newEnemy->hp = hp;
            newEnemy->hp_Given = hp;
            // Link another end of newEnemy to NULL.
            newEnemy->next = NULL;

            // If there is nothing in the enemies section yet.
            if (current->enemies == NULL) {
                // Link the newEnemy under the tower.
                current->enemies = newEnemy;
            } else { // These is existing enemy in enemies section.
                       ^
                       + =================================================================== +
                       + INLINE COMMENTS: it's usually better for comments to go on the    +
                       + line above                                                        +
                       + =================================================================== +
                // Loop thro enemies until current->enemies->next == NULL.
                currentEnemy = current->enemies;
                while (currentEnemy->next != NULL) {
                    currentEnemy = currentEnemy->next;
                    ^
                    + =================================================================== +
                    + OVERDEEP_NESTING: nesting too deep: 5 (try moving some of the     +
                    + logic into another function)                                      +
                    + =================================================================== +
                }
                // Inserting newEnemy into list.
                currentEnemy->next = newEnemy;

            }
            // Break the program, so it will not continue to scan the same name
            // for the rest of the linked list.
            break;
               ^
               + ================= +
               + Avoid using break +
               + ================= +
        }
        current = current->next;
        if (current == NULL) {
            return ERROR_NO_LOCATION;
        }
    }
    return SUCCESS;
```

```c
}

// Add a new tower to the realm by looping through the list, and insert it
// after previous position.
// The function take realm, previous position, the new name of tower,
// its power and uses as input.
// The function will return SUCCESS(0) if new tower if successfully created.
int new_tower(Realm realm, char *prev_name, char *name, int power, int uses) {
    // Ensure that the stats you have been given for power
    // and uses are not below 1. If they are, return ERROR_INVALID_STAT.
                                                 ^
                                                 + ========================== +
                                                 + Avoid the self explanatory +
                                                 + ========================== +
    if (power < 1 || uses < 1) {
        return ERROR_INVALID_STAT;
    }

    // Create a pointer current, used to loop thro the list,
    // in order to match the same name.
    struct location *current = realm->castle;
    while (current != NULL) {
        // If current location is same name as required.
        if (strcmp(prev_name, current->name) == 0) {
                 ^
                 + ============================================ +
                 + Could this go in a function find_location()? +
                 + ============================================ +
            // Place a new tower, with the given stats,
            // directly after the location which you found above.
            // This tower is inserted into the linked list,
            // which adds an element and doesn't replace the Location prev_name.

            // Create a new location for new tower.
            struct location *newTower = new_location(name);
            // Assign name, power and uses to new tower.
            strcpy(newTower->name, name);
            newTower->power = power;
            newTower->uses = uses;
            // For stage 5, effect of EFFECT_NONE added to all new tower.
            newTower->effect = EFFECT_NONE;
            // Assign the new tower into desired position.
            newTower->next = current->next;
            current->next = newTower;
            // Break the program, so it will not continue to scan the same name
            // for the rest of the linked list.
            break;
        }

        current = current->next;
        // If current reaches the NULL after the move, return error.
        if (current == NULL) {
            return ERROR_NO_LOCATION;
        }

    }

    // Return SUCCESS to indicate success.
    return SUCCESS;
}

////////////////////////////////////////////////////////////////////////
//                          Stage 3 Functions                         //
////////////////////////////////////////////////////////////////////////

// Destroy the realm, and free any associated memory by looping through
// all the links including both tower/land/castle/lair and enemies.
// The function take realm as input,
// there is no output as this is void type function.
void destroy_realm(Realm realm) {
    // Free the enemies first, then locations, and lastly castle and lair.
    // Create two pointers, one "current" to loop thro the linked list,
    // another "privious" pointing to the memory will be free.
    struct location *current = realm->castle;

    // Create two while loop to scan thro each enemy in each location.
    // Free location as decleared in new_location.
    while (current != NULL) {
        struct location *previous = NULL;
        struct enemy *currentEnemy = current->enemies;
        // Free enemies as decleared in new_enemy.
        while (currentEnemy != NULL) {
            struct enemy *previousEnemy = NULL;

            previousEnemy = currentEnemy;
            currentEnemy = currentEnemy->next;
            free(previousEnemy);
        }
        previous = current;
        current = current->next;
        free(previous);
    }
    // At the end, free realm as decleared in Realm new_realm(void).
    free(realm);
}



/*
Advance enemies towards the castle.
Advance_enemies will go through the realm,
moving each enemy from their current Location to
the Location above in the linked list.
The function take realm as input,
the number of enemies who pass the castle will be recorded as output.
*/
int advance_enemies(Realm realm) {
    // Create a pointer "current", started at Castle
    // (We could started From Lair, but required to inverted the link),
    // used to loop thro the linked list, and move the enemies forward.
    struct location *current = realm->castle;
    // Create a pointer "previous", started at NULL,
    // previous always move one step slower than current,
    // in order to reference current to pass enemy forward.
    struct location *previous = NULL;
    // Create a counter in record the number of enemies have been removed.
    int counter = 0;

    // Loop thro linked list until NULL is reached.
    while (current != NULL) {
        // Check whether the enemy section is empty under current location.
        // If empty, current move on.
        // Otherwise, change the link to previous location,
        // if current is castle, then previous is NULL.
        if (current->enemies != NULL) {
            // Create a new pointer to loop thro until NULL is reached.
            struct enemy *currentEnemy = current->enemies;
            struct enemy *previousEnemy = current->enemies;
            if (strcmp(current->name, "Castle") == 0) {
                // Check the number of enemies in the castle.
                while (currentEnemy != NULL) {
                    // Number of enemies pass thro castle.
                    counter++;
                    ^
                    + ================================================================== +
                    + OVERDEEP_NESTING: nesting too deep: 5 (try moving some of the     +
                    + logic into another function)                                       +
                    + ================================================================== +
                    currentEnemy = currentEnemy->next;
                }
                // Then remove all these enemies from the map.
                current->enemies = NULL;

            }
            // For stage 5, if tower with effect of ice has been detected.
            else if (current->power > 0 && current->uses > 0 &&
            current->effect == EFFECT_ICE) {
                // Determine the enemy who has HP less than current's power.
                while (currentEnemy != NULL) {
                    // If current enemy has hp less than tower's power,
                    // then it will be frozen under this tower.
                    // Hence no further action is required other than move pass.
                    if (currentEnemy->hp < current->power) {
                        // Move pass it.
                        previousEnemy = currentEnemy;
                        ^
                        + ================================================================ +
                        + OVERDEEP_NESTING: nesting too deep: 6 (try moving some of the     +
                        + logic into another function)                                     +
                        + ================================================================ +
                        currentEnemy = currentEnemy->next;
                    }
                    // If the first enemy under the tower has hp > power,
                    // who cannot limited to ice effect.
                    // It will advance to next tower as normal.
                    else if (currentEnemy->hp >= current->power &&
                    currentEnemy == current->enemies)
                    {
                        current->enemies = currentEnemy->next;
                        currentEnemy->next = previous->enemies;
                        previous->enemies = currentEnemy;
                        currentEnemy = current->enemies;
                    }
                    // If enemies are not in the two categories above,
                    // then advance to next tower as normal.
                    else {
                        previousEnemy->next = currentEnemy->next;
                        currentEnemy->next = previous->enemies;
                        previous->enemies = currentEnemy;
                        currentEnemy = previousEnemy->next;
                    }
                }
            }
            // Enemies will advance to next tower normally
            // as no effect is applied.
            else {
                while (currentEnemy != NULL) {
                    current->enemies = currentEnemy->next;
                    ^
                    + ================================================================ +
                    + OVERDEEP_NESTING: nesting too deep: 5 (try moving some of the     +
                    + logic into another function)  (+ 2 other overdeep_nesting errors) +
                    + ================================================================ +
                    currentEnemy->next = previous->enemies;
                    previous->enemies = currentEnemy;
                    currentEnemy = current->enemies;
                }
            }
        }
        // Move previous to the current position.
        previous = current;
        // Then move the current to next position in the linked list.
        current = current->next;
    }

    // Function for portal effect below, and reset the pointers.
    // Since effect portal only applied when the movement is finished,
    // hence the portal effect function is created after the advance function
    // is completed.
    current = realm->castle;
    while (current != NULL) {
        struct enemy *currentEnemy = current->enemies;
        if (current->power > 0 && current->uses > 0 &&
        current->effect == EFFECT_PORTAL)
        {
            // Move all the enemies under portal effect tower back to lair.
            while (currentEnemy != NULL) {
                current->enemies = currentEnemy->next;
                currentEnemy->next = realm->lair->enemies;
                realm->lair->enemies = currentEnemy;
                currentEnemy = current->enemies;
                // Convert the portal effect back to none effect,
                // after the portal effect has been applied once.
                current->effect = EFFECT_NONE;
            }

        }
        // This is sorting function, more detailed see function description.
        // which sorts enemies alphabetically under the same location.
        // As all the effects and re-arrangements are completed,
        // we should sort the enemies alphabetically to conclude the function.
        if (current->enemies != NULL) {
            sorting(current);
        }

        current = current->next;
    }
    return counter;
}



// Apply damage from the enemies at each tower to that tower.
// Go through each Enemy at that Tower, and reduce it's HP by the Tower's power.
// After the forward movements have been done to enemies,
// apply damage to enemies, and reduce use by one to tower.
// This function take realm as input, the number of damaged enemies as outputs.
int apply_damage(Realm realm) {
    // Initiate a counter to record the number of enemies damaged this way.
    int counter = 0;
    // Create a pointer "current" to loop thro locations from castle till NULL.
    struct location *current = realm->castle;

    while (current != NULL) {
        // Proceed if current location is tower.
        // Tower is when it has both non-zero power and uses.
        if (current->uses != 0 && current->power != 0 &&
        current != realm->castle) {
            // Check the number of enemies in the location.
            // Create a new pointer to loop thro until NULL is reached.
            struct enemy *currentEnemy = current->enemies;

            // Apply damage to all enemies under the tower.
            while (currentEnemy != NULL) {
                // Apply damage to current enemy.
                currentEnemy->hp = (currentEnemy->hp) - current->power;
                // Reduce tower's use by one.
                current->uses--;
                // The number of damaged enemy increase by one.
                counter++;
                // Move on.
                currentEnemy = currentEnemy->next;
            }
            remove_enemy(current, currentEnemy);

        }
        else if (current == realm->castle) { // When enemy is under castle.
                                               ^
                                               +
================================================================= +
                                               + INLINE COMMENTS: it's usually better for comments to go on
the    +
                                               + line above
+
                                               +
================================================================= +
            // Enemies can cause castle's defense to decrease.
            // Check the number of enemies in the castle.
```

```
                    // Create a new pointer to loop thro until NULL is reached.
                    struct enemy *currentEnemy = current->enemies;
                    while (currentEnemy != NULL) {
                        // Castle's remaining health.
                        realm->castle->defense -= currentEnemy->hp;
                        currentEnemy = currentEnemy->next;
                    }

                }
                current = current->next;
            }

            return counter;
        }

        ////////////////////////////////////////////////////////////////////////
        //                         Stage 4 Functions                          //
        ////////////////////////////////////////////////////////////////////////

        /*
        The function of "apply_buff" apply the specified buff
        to the relevant towers or enemies based on search term and buff type.
        This funtion take realm, search term, buff type, and amount of buff as inputs.
        The number of buff will be applied are the outputs of this function.
        */
        int apply_buff(Realm realm, char *search_term, Buff buff, int amount){
            // Record number of buff has been applied.
            int counterForBuff = 0;
            // Loop through linked list, to determine amount of
            // towers or enemies are matching the search terms.
            // Based on the effect type, different function can be achieved.
            // Create a pointer "current" to matching cases.
            struct location *current = realm->castle;
            while (current != NULL) {
                // Seperate tower and enemies into two different types.
                if (current->power != 0 && current->uses != 0 &&
                buff != BUFF_ENEMY_HP &&
                search_function_tower(current, search_term) == 1) { // Tower
                                                                      ^
                                                                      +
================================================================= +
                                                                      + INLINE COMMENTS: it's usually better for
comments to go on the    +
                                                                      + line above
+
                                                                      +
================================================================= +
                    // Apply buff.
                    if (buff == BUFF_TOWER_POWER) {
                        // Increase each of their power by the specified amount.
                        current->power += amount;
                        // Return the number of towers you found this way.
                        counterForBuff++;
                    } else if (buff == BUFF_TOWER_USES) {
                        // Increase each of their uses by the specified amount.
                        current->uses += amount;
                        counterForBuff++;
                    }

                } else if (current->enemies != NULL) { // Enemies.
                    // Create a pointer "currentEnemy" to matching cases.
                    struct enemy *currentEnemy = current->enemies;
                    // Loop thro each enemy under the location until NULL is reached.
                    while (currentEnemy != NULL) {
                        // Apply buffs to enemy.
                        if (buff == BUFF_ENEMY_HP &&
                        search_function_enemy(currentEnemy, search_term) == 1) {
                            // Increase each of their HP by the specified amount.
                            currentEnemy->hp += amount;
                            ^
                            + ================================================================= +
                            + OVERDEEP_NESTING: nesting too deep: 5 (try moving some of the     +
```

```c
                    + logic into another function)                                 +
                    + ================================================================== +
                    // Return the number of enemies you found this way.
                    counterForBuff++;
                }
                currentEnemy = currentEnemy->next;
            }
            // If debuff has been applied,
            // enemies' hp has decreased, then remove enemy.
            remove_enemy(current, currentEnemy);
        }
        current = current->next;
    }
    return counterForBuff;
}


////////////////////////////////////////////////////////////////////////
//                          Stage 5 Functions                         //
////////////////////////////////////////////////////////////////////////

/*
Apply the effect described by effect_type to every tower matching search_term.
The function take realm, search term, and effect type as inputs,
the number of towers that match search term will be returned.
*/
int apply_effect(Realm realm, char *search_term, Effect effect) {
    // Record number of towers that have been matched.
    int counterForMatchedTower = 0;
    // Loop through linked list, to determine amount of
    // towers are matching the search terms.
    // Based on the effect type, different function can be achieved.
    // Create a pointer "current" to matching cases.
    struct location *current = realm->castle;
    while (current != NULL) {
        // Locate the towers matching search term.
        if (current->power > 0 && current->uses > 0 &&
        search_function_tower(current, search_term)) {
            // Then determine different effect.
            if (effect == EFFECT_ICE)
            {
                // If an Enemy would move from the Tower with this effect
                // to the next Location,
                // and that enemy has HP less than to the current tower's power,
                // it stays at the current Tower.

                // Assign effect to the tower.
                current->effect = EFFECT_ICE;
                counterForMatchedTower++;
            }
            else if (effect == EFFECT_PORTAL)
            {
                /*
                After all enemies have moved,
                if there are enemies at portal towers,
                they should all be moved back to the Lair.
                This effect is removed from a tower after
                it has finished moving enemies.
                */
                // Assign effect to the tower.
                current->effect = EFFECT_PORTAL;
                counterForMatchedTower++;
            }
        }
        current = current->next;
    }
    return counterForMatchedTower;
}


////////////////////////////////////////////////////////////////////////
//                        Self-Created Functions                      //
```

```c
////////////////////////////////////////////////////////////////////////

/*
The function of "remove_enemy" removes the enemy who has hp not above zero.
The function takes current location (tower/land), and currentEnemy as inputs.
It will scan through every single enemies under the location,
and remove it if hp < 0 is detected (free node), then re-do the process,
until all enemies have hp > 0.
This function is void type, hence no output will be produced.
*/
static void remove_enemy(struct location *current, struct enemy *currentEnemy) {
    // Remove enemies.
    // Reset the currentEnemy, loop thro the enemies again,
    // in order to check any enemies'hp are below 0.
    currentEnemy = current->enemies;
    // Create a pointer "previousEnemy",
    // which is move one step slower than currentEnemy,
    // in order to link the previousEnemy and currentEnemy->next,
    // when currentEnemy is removed.
    struct enemy *previousEnemy = NULL;
    while (currentEnemy != NULL) {
        // Check whether hp is below 0.
        if (currentEnemy->hp <= 0) {
            // If this is the first enemy under the tower.
            if (current->enemies == currentEnemy) {
                current->enemies = currentEnemy->next;
                free(currentEnemy);
                // Reset currentEnemy back to head.
                currentEnemy = current->enemies;
            } else { // Else this is not the first enemy under.
                       ^
                       + ================================================================= +
                       + INLINE COMMENTS: it's usually better for comments to go on the    +
                       + line above                              (+ 1 other inline         +
                       + comments errors)                                                  +
                       + ================================================================= +
                previousEnemy->next = currentEnemy->next;
                free(currentEnemy);
                // Reset.
                currentEnemy = current->enemies;
                previousEnemy = NULL;
            }
        } else { // hp > 0, move on.
            previousEnemy = currentEnemy;
            currentEnemy = currentEnemy->next;
        }

    }
}

/*
Search function for tower and enemy by looping through each elements,
and conduct required searches.
These search functions are implementing prefix search, not exact match.
which means the word can pass the test as long as
the search term can match the word for the same length.
Both functions take current location (either tower or enemy),
and required search term as inputs.
They will return 1 if matching, otherwise return 0.
These two functions are basically have the same content, but different input,
search_function_tower takes struct location type of current as input,
search_function_enemy takes struct enemy type of current as input.
   ^
   + ====================== +
   + Great function comment +
   + ====================== +
*/
static int search_function_tower (struct location *current, char *search_term) {
    // Initiate a counter for counting each word in current.
    int counter = 0;
    // Counting each word for search term.
    int counterForRange = 0;
    // Prefix search:
    // Search will finished if looped thro search term.
    // Return 0 if any character is not matching,
    // return 1 only if all search terms are exactly matching.
    while (counterForRange < strlen(search_term)) {

        // If no [] is detected, match each character.
        if (search_term[counterForRange] != '[' ) {
            if (search_term[counterForRange] == current->name[counter]) {
                // matched, move on to next character.
                  ^
                  + ========================================== +
                  + Avoid empty conditions - invert the logic  +
                  + instead                                    +
                  + ========================================== +
            } else {
                // Not matched, end the function.
                return 0;
            }
        } else if (search_term[counterForRange] == '[') {
            // Each character will have a unique matching number,
            // If it is still 1 at the end of the
            int matching = 0;
            // Then we need to find the location of ].
            while (search_term[counterForRange] != ']') {
                // For single matching stype input. e.g.[abc]
                if (search_term[counterForRange] == current->name[counter]) {
                    // Then matched.
                    matching = 1;
                    ^
                    + ================================================================ +
                    + OVERDEEP_NESTING: nesting too deep: 5 (try moving some of the     +
                    + logic into another function)                                      +
                    + ================================================================ +
                }
                // For range style input e.g.[a-z]
                else if (search_term[counterForRange] == '-') {
                    if (search_term[counterForRange - 1] <=
                    current->name[counter]
                    && search_term[counterForRange + 1] >=
                    current->name[counter]) {
                        // Matched
                        matching = 1;
                        ^
                        + ================================================================ +
                        + OVERDEEP_NESTING: nesting too deep: 6 (try moving some of the     +
                        + logic into another function)                                      +
                        + ================================================================ +
                    }
                }
                counterForRange++;

            }
            // If the programme didnt found corresponded characters,
            // at the end of ]
            // Which means no matched.
            if (matching != 1) {
                return 0;
            }
        }
        counter++;
        counterForRange++;
    }

    return 1;
}

// Search function for enemy.
// Return 1 if completely matched.
// Detailed description is above.
static int search_function_enemy (struct enemy *current, char *search_term) {
    // Initiate a counter for counting each word in current.
    int counter = 0;
    // Counting each word for search term.
    int counterForRange = 0;
    // Prefix search:
    // Search will finished if looped thro search term.
    // Return 0 if any character is not matching,
    // return 1 only if all search terms are exactly matching.
    while (counterForRange < strlen(search_term)) {

        // If no [] is detected, match each character.
        if (search_term[counterForRange] != '[' ) {
            if (search_term[counterForRange] == current->name[counter]) {
                // matched, move on to next character.
            } else {
                // Not matched, end the function.
                return 0;
            }
        } else if (search_term[counterForRange] == '[') {
            // Each character will have a unique matching number,
            // If it is still 1 at the end of the
            int matching = 0;
            // Then we need to find the location of ].
            while (search_term[counterForRange] != ']') {
                // For single matching stype input. e.g.[abc]
                if (search_term[counterForRange] == current->name[counter]) {
                    // Then matched.
                    matching = 1;
                    ^
                    + ================================================================= +
                    + OVERDEEP_NESTING: nesting too deep: 5 (try moving some of the     +
                    + logic into another function)                                      +
                    + ================================================================= +
                }
                // For range style input e.g.[a-z]
                else if (search_term[counterForRange] == '-') {
                    if (search_term[counterForRange - 1] <=
                    current->name[counter]
                    && search_term[counterForRange + 1] >=
                    current->name[counter]) {
                        // Matched
                        matching = 1;
                        ^
                        + ================================================================= +
                        + OVERDEEP_NESTING: nesting too deep: 6 (try moving some of the     +
                        + logic into another function)                                      +
                        + ================================================================= +
                    }
                }
                counterForRange++;

            }
            // If the programme didnt found corresponded characters,
            // at the end of ]
            // Which means no matched.
            if (matching != 1) {
                return 0;
            }
        }
        counter++;
        counterForRange++;
    }

    return 1;
}


/*
The function "sorting" sorts all enemies alphabetically based on their names
under the same location. This function is achieved similar to the
bubble sort technique, which compares two variables, then swap, move to the
next position, and then repeat until all variables are in alphabetical order.
The function take struct location * type as input, and no output based on its
```

```c
void type.
*/
static void sorting(struct location *current) {
    struct enemy *previousEnemy = current->enemies;
    struct enemy *currentEnemy = current->enemies;
    struct enemy *nextEnemy = current->enemies->next;
    while (nextEnemy != NULL) {
        while (strcmp(currentEnemy->name, nextEnemy->name) > 0) {
            // Z > A.
            // First number
            if (currentEnemy == current->enemies) {
                currentEnemy->next = nextEnemy->next;
                nextEnemy->next = current->enemies;
                current->enemies = nextEnemy;
                // reset
                previousEnemy = current->enemies;
                currentEnemy = current->enemies;
                nextEnemy = current->enemies->next;

            }
            else {// Assuming in the middle of role.
                currentEnemy->next = nextEnemy->next;
                previousEnemy->next = nextEnemy;
                nextEnemy->next = currentEnemy;
                // reset
                previousEnemy = current->enemies;
                currentEnemy = current->enemies;
                nextEnemy = current->enemies->next;
            }

        }
        previousEnemy = currentEnemy;
        currentEnemy = nextEnemy;
        nextEnemy = nextEnemy->next;
    }

}




////////////////////////////////////////////////////////////////////////
//                      Provided print functions                      //
//         NOTE: YOU SHOULD NOT MODIFY THE FOLLOWING FUNCTIONS        //
////////////////////////////////////////////////////////////////////////

void print_tower(char *name, int power, int uses, int effect) {
    printf(" ^ %32s [pow: %3d | uses: %3d]", name, power, uses);
    if (effect == EFFECT_NONE) printf(" {%c}", EFFECT_NONE_CHAR);
    if (effect == EFFECT_PORTAL) printf(" {%c}", EFFECT_PORTAL_CHAR);
    if (effect == EFFECT_ICE) printf(" {%c}", EFFECT_ICE_CHAR);
    printf("\n");
}

void print_land(char *name) {
    printf(" ^ %32s [____________________]\n", name);
}

void print_castle(char *name, int defense) {
    printf(" ^ %32s [Castle Defenses: %3d]\n", name, defense);
}

void print_enemy(char *name, int cur_hp, int max_hp) {
    printf(" ^ %40s [hp: %d/%d]\n", name, cur_hp, max_hp);
}

////////////////////////////////////////////////////////////////////////
//                 End of provided print functions                    //
////////////////////////////////////////////////////////////////////////



```

```c
// Assignment 2 20T2 COMP1511: Castle Defense
//
// This program was written by Zheng Luo (z5206267@ad.unsw.edu.au)
// on July/2020
//
// Version 1.0.0 (2020-07-20): Assignment released.
// Version 1.0.1 (2020-07-21): Add return value to main.

#include <stdio.h>
#include <string.h>
#include <assert.h>

#include "realm.h"
#include "test_realm.h"
#include "capture.h"

////////////////////////////////////////////////////////////////////
//                        Function Protoypes                      //
////////////////////////////////////////////////////////////////////

// TODO: Add your own prototypes here, if necessary.

////////////////////////////////////////////////////////////////////
//                         Helper Functions                       //
////////////////////////////////////////////////////////////////////

// Find the string 'needle' in 'haystack'
int string_contains(char *haystack, char *needle) {
    return strstr(haystack, needle) != NULL;
}

// NOTE: You should not declare any structs from realm.c here.
// Declaring structs from realm.c in this function is against the
// rules of the assignment.

// Main function: this function will not be used in testing, you do not
// need to change it.
// If you want to write your own tests, add them in `extra_tests`.
int main(int argc, char *argv[]) {
    printf("\n================== Castle Defense Tests ==================\n");

    test_add_location();
    test_print_realm();
    test_new_enemy();
    test_new_tower();
    test_apply_damage();
    extra_tests();

    return 0;
}

// NOTE: These tests are explained in `test_realm.h`

void test_add_location(void) {
    printf("Test whether this add_location follows the spec: ");

    // Test 1: Does add_location's return value count the Castle & Lair?

    Realm test_realm = new_realm();

    int num_locations = add_location(test_realm, "Location");
    if (num_locations != 3) {
        printf(DOES_NOT_FOLLOW_SPEC);
        // since we don't want to print FOLLOWS_SPEC at the end,
        // we just return to the main function here.
        return;
    }

    // Test 2: Does add_location return the correct amount for lots of locations?
    // Test if 1024 locations will be added.
    Realm test_realm_test2 = new_realm();
    char string[1024];
    num_locations = 0;
    int counter = 0;
    while (counter < 1024) {
        // store difference names into string based on counter.
        sprintf(string, "Location%d", counter);
        // the name of location will be based on string.
        num_locations = add_location(test_realm_test2, string);
        counter++;
    }
    if (num_locations != 1026) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }

    // Test 3: Add your own test, and explain it below:
    // Test whether added location is located before lair.
    Realm test_realm_test3 = new_realm();
    char string3[1024];
    char expected_string[1024] =
    " ^                           Castle [Castle Defenses: 100]\n\
 ^                       Location1 [____________________]\n\
 ^                       Location2 [____________________]\n\
 ^                            Lair [____________________]\n";
    // Two locations will be added in test 3,
    add_location(test_realm_test3, "Location1");
    add_location(test_realm_test3, "Location2");
    // the location2 must located before lair and after location1.
    // castle->location1->location2->lair
    CAPTURE(print_realm(test_realm_test3), string3, 1024);
    if (strcmp(string3, expected_string) != 0) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }


    printf(FOLLOWS_SPEC);
}

void test_print_realm(void) {
    printf("Test whether this print_realm follows the spec: ");

    // Test 1: Does print_realm show the Castle?

    Realm test_realm = new_realm();

    char print_text[10000];
    CAPTURE(print_realm(test_realm), print_text, 10000);

    if (!string_contains(print_text, "Castle")) {
        printf(DOES_NOT_FOLLOW_SPEC);
        // since we don't want to print FOLLOWS_SPEC at the end,
        // we just return to the main function here.
        return;
    }

    // Test 2: Does print_realm show the correct HP of the castle?

    Realm test2_realm = new_realm();

    char print_text_test2[10000];
    CAPTURE(print_realm(test2_realm), print_text_test2, 10000);

    if (!string_contains(print_text_test2, "Castle Defenses: 100")) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }

    // Test 1: Does print_realm show the Lair?

    Realm test3_realm = new_realm();

    char print_text3[10000];
    CAPTURE(print_realm(test3_realm), print_text3, 10000);

    if (!string_contains(print_text, "Lair")) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }

    printf(FOLLOWS_SPEC);
}

// Stage 2

void test_new_enemy(void) {
    printf("Test whether this new_enemy follows the spec: ");

    // Test 1: Does new_enemy work if you're adding to the Lair?
    Realm test_realm = new_realm();
    // Generate a tested enemy call "Roger" with 5hp at Lair.
    // This is following the spec, if return value is SUCCESS.
    if (new_enemy(test_realm, "Lair" , "Roger", 5) != SUCCESS) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }

    // Test 2: Does new_enemy work if you're adding to the Castle?
    Realm test2_realm = new_realm();
    // Generate a tested enemy call "Roger" with 5hp at Castle.
    // This is following the spec, if return value is SUCCESS.
    if (new_enemy(test2_realm, "Castle" , "Roger", 5) != SUCCESS) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }
    printf(FOLLOWS_SPEC);
}

void test_new_tower(void) {
    printf("Test whether this new_tower follows the spec: ");

    // Test 1: Does new_tower work if you're adding after another tower?
    Realm test_realm = new_realm();
    // Create first test tower right after the Castle with power and uses 5.
    new_tower(test_realm, "Castle", "TestTower1", 5, 5);
    // Then create second test tower after test tower 1.
    // If second tower does not return success, then print "NOT FOLLOW SPEC"
    if (new_tower(test_realm, "TestTower1", "TestTower2", 5, 5) != SUCCESS) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }

    // Test 2: Does new_tower work if it's power or uses are less than 1?
    Realm test2_realm = new_realm();
    // Create a tower with power 0, uses 0.
    // if it return success, then print "NOT FOLLOW SPEC".
    if (new_tower(test2_realm, "Castle", "TestTower", 0, 0) == SUCCESS) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }

    // Test 3: Does new_tower work if there is no previous location
    // in the link as given in the function?
    Realm test3_realm = new_realm();
    // Create a tower after a non-existing location.
    // If it return success, then print "NOT FOLLOW SPEC".
    if (new_tower(test3_realm, "NON_EXISTING LOCATION",
        "TestTower", 5, 5) == SUCCESS) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }

    printf(FOLLOWS_SPEC);
}
```

```c

// Stage 3 (2 marks)

void test_apply_damage(void) {

    printf("Test whether this apply_damage follows the spec: ");

    // Test 1: Does apply_damage actually destroy enemies?
    Realm test_realm = new_realm();
    char string[1024];


    // Create a tower and a enemy under the tower,
    // with a hp lower than tower's power.
    new_tower(test_realm, "Castle", "TestTower", 5, 5);
    new_enemy(test_realm, "TestTower", "TestEnemy", 3);
    // Apply damage
    apply_damage(test_realm);
    CAPTURE(print_realm(test_realm), string, 1024);

    if (string_contains(string, "TestEnemy")) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }


    // Test 2: Could apply damage function destory and damage multiple enemies
    // under the same location at the same time?
    Realm test2_realm = new_realm();
    char string2[1024];
    // Create a tower and multiple enemies under the tower, with different hp.
    new_tower(test2_realm, "Castle", "TestTower", 5, 5);
    new_enemy(test2_realm, "TestTower", "TestEnemy1", 3);
    new_enemy(test2_realm, "TestTower", "TestEnemy2", 10);
    new_enemy(test2_realm, "TestTower", "TestEnemy3", 3);
    // Apply damage
    apply_damage(test2_realm);
    CAPTURE(print_realm(test2_realm), string2, 1024);
    if (string_contains(string2, "TestEnemy1") ||
    string_contains(string2, "TestEnemy3")) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }

    printf(FOLLOWS_SPEC);
}

// Stage 4 (1 marks)

void test_apply_buff(void) {
    printf("Test whether this apply_buff follows the spec: ");
    // Test 1: Does the buff has been successfully added?
    // Apply 5 power buff to TestTower, if the function is not returning 1
    // then print DOES NOT FOLLOW SPEC.
    Realm test_realm = new_realm();
    new_tower(test_realm, "Castle", "TestTower", 5, 5);
    if (apply_buff(test_realm, "TestTower", BUFF_TOWER_POWER, 5) != 1) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }

    // Test 2: Does the buff has been added correctly?
    // Apply 5 power buff to TestTower with 5 power originally,
    // if it is not returning 10 as buff, then return DOES NOT FOLLOW SPEC.
    Realm test2_realm = new_realm();
    char string[1024];
    new_tower(test2_realm, "Castle", "TestTower", 5, 5);
    apply_buff(test2_realm, "TestTower", BUFF_TOWER_POWER, 5);
    CAPTURE(print_realm(test2_realm), string, 1024);
    if (!string_contains(string, "TestTower [pow:  10 | uses:   5] {n}")) {
        printf(DOES_NOT_FOLLOW_SPEC);
        return;
    }

    printf(FOLLOWS_SPEC);
}

  ^
  + ================================================================= +
  + NOTE: broken indentation starts here -- watch out for stray       +
  + curly brackets, or functions with ';' not '{'                     +
  + ================================================================= +
// Extra Tests (not worth marks, but fun!)

void extra_tests(void) {
    // TODO: add extra tests, if you'd like to.
}
```

Style Summary

```
^
+ ========================================================================================= +
+                                                                                           +
+ ========================================                                                  +
+ =====    Style feedback summary:   =====                                                  +
+ ========================================                                                  +
+                                                                                           +
+ ====== Header Comment =====                                                               +
+ Header comment has 6 lines (1 lines of description)                                       +
+ Header comment contains zID!                                                              +
+                                                                                           +
+ ====== #defines =====                                                                     +
+ No additional constants #defined                                                          +
+                                                                                           +
+ ====== Nesting Depth =======                                                              +
+ Nesting depth was too much: max depth of 6!                                               +
+                                                                                           +
+ ====== Whitespace Errors =====                                                            +
+ No whitespace errors!                                                                     +
+                                                                                           +
+ ====== Indentation =======                                                                +
+ Indentation analysis disabled due to invalid/non-compiling C code                         +
+                                                                                           +
+ ====== Over-long Lines =======                                                            +
+ No lines over 80 characters!                                                              +
+                                                                                           +
+ ====== Complex If Statements =======                                                      +
+ No complex if statements!                                                                 +
+                                                                                           +
+ ====== Functions and Prototypes =====                                                     +
+                                                                                           +
+ Function implementations:                                                                 +
+                                                                                           +
+ realm.c functions:                                                                        +
+     68  Realm new_realm(void) {                                                           +
+          -> 17 lines long (15 code lines)                                                 +
+          -> (has no function comment!)                                                    +
+     89  static Location new_location(char *name) {                                         +
+          -> 13 lines long (13 code lines)                                                 +
+          -> (has a 1 line function comment)                                               +
+    110  int add_location(Realm realm, char *name) {                                       +
+          -> 33 lines long (30 code lines)                                                 +
+          -> (has a 1 line function comment)                                               +
+    149  void print_realm(Realm realm) {                                                   +
+          -> 38 lines long (33 code lines)                                                 +
+          -> (has a 4 line function comment)                                               +
+    197  int new_enemy(Realm realm, char *location_name, char *name, int hp) {             +
+          -> 51 lines long (48 code lines)                                                 +
+          -> (has a 4 line function comment)                                               +
+    255  int new_tower(Realm realm, char *prev_name, char *name, int power, int uses) {   +
+          -> 44 lines long (39 code lines)                                                 +
+          -> (has a 5 line function comment)                                               +
+    309  void destroy_realm(Realm realm) {                                                 +
+          -> 25 lines long (23 code lines)                                                 +
+          -> (has a 4 line function comment)                                               +
+    346  int advance_enemies(Realm realm) {                                                +
+          -> 117 lines long (112 code lines)                                               +
+          -> (has a 7 line function comment)                                               +
+    472  int apply_damage(Realm realm) {                                                   +
+          -> 45 lines long (40 code lines)                                                 +
+          -> (has a 5 line function comment)                                               +
+    529  int apply_buff(Realm realm, char *search_term, Buff buff, int amount){            +
+          -> 47 lines long (46 code lines)                                                 +
+          -> (has a 5 line function comment)                                               +
+    588  int apply_effect(Realm realm, char *search_term, Effect effect) {                 +
+          -> 41 lines long (34 code lines)                                                 +
+          -> (has a 4 line function comment)                                               +
+    644  static void remove_enemy(struct location *current, struct enemy *currentEnemy) { +
+          -> 32 lines long (31 code lines)                                                 +
+          -> (has a 7 line function comment)                                               +
+    691  static int search_function_tower (struct location *current, char *search_term) { +
+          -> 55 lines long (52 code lines)                                                +
+          -> (has a 12 line function comment)                                             +
+    751  static int search_function_enemy (struct enemy *current, char *search_term) {    +
+          -> 55 lines long (52 code lines)                                                +
+          -> (has a 3 line function comment)                                              +
+    817  static void sorting(struct location *current) {                                  +
+          -> 34 lines long (31 code lines)                                                +
+          -> (has a 7 line function comment)                                              +
+    861  void print_tower(char *name, int power, int uses, int effect) {                  +
+          -> 6 lines long (6 code lines)                                                  +
+          -> (has no function comment!)                                                   +
+    869  void print_land(char *name) {                                                    +
+          -> 2 lines long (2 code lines)                                                  +
+          -> (has no function comment!)                                                   +
+    873  void print_castle(char *name, int defense) {                                     +
+          -> 2 lines long (2 code lines)                                                  +
+          -> (has no function comment!)                                                   +
+    877  void print_enemy(char *name, int cur_hp, int max_hp) {                           +
+          -> 2 lines long (2 code lines)                                                  +
+          -> (has no function comment!)                                                   +
+                                                                                          +
+ test_realm.c functions:                                                                  +
+     28  int string_contains(char *haystack, char *needle) {                              +
+          -> 2 lines long (2 code lines)                                                  +
+          -> (has no function comment!)                                                   +
+     39  int main(int argc, char *argv[]) {                                               +
+          -> 11 lines long (9 code lines)                                                 +
+          -> (has no function comment!)                                                   +
+     54  void test_add_location(void) {                                                   +
+          -> 55 lines long (48 code lines)                                                +
+          -> (has no function comment!)                                                   +
+    111  void test_print_realm(void) {                                                    +
+          -> 42 lines long (29 code lines)                                                +
+          -> (has no function comment!)                                                   +
+    157  void test_new_enemy(void) {                                                      +
+          -> 21 lines long (19 code lines)                                                +
+          -> (has no function comment!)                                                   +
+    180  void test_new_tower(void) {                                                      +
+          -> 35 lines long (31 code lines)                                                +
+          -> (has no function comment!)                                                   +
+    219  void test_apply_damage(void) {                                                   +
+          -> 42 lines long (34 code lines)                                                +
+          -> (has no function comment!)                                                   +
+    265  void test_apply_buff(void) {                                                     +
+          -> 23 lines long (23 code lines)                                                +
+          -> (has no function comment!)                                                   +
+    295  void extra_tests(void) {                                                         +
+          -> 1 lines long (2 code lines)                                                  +
+          -> (has no function comment!)                                                   +
+                                                                                          +
+ ====== Variables =======                                                                 +
+ Declared 53 additional variables:                                                        +
+     28  int power;                                                                       +
+     29  int uses;                                                                        +
+     30  int defense;                                                                     +
+     41  int hp;                                                                          +
+     42  int hp_Given;                                                                    +
+     91  struct location *place = malloc(sizeof(struct location));                        +
+    112  struct location *newLand = new_location(name);                                   +
+    116  int counter = 4;                                                                 +
+    132  struct location *current = realm->castle->next;                                  +
+    151  struct location *current = realm->castle;                                        +
+    171  struct enemy *currentEnemy = current->enemies;                                   +
+    207  struct location *current = realm->castle;                                        +
+    213  struct enemy *newEnemy = malloc(sizeof(struct enemy));                           +
+    216  struct enemy *currentEnemy = NULL;                                               +
+    264  struct location *current = realm->castle;                                        +
+    274  struct location *newTower = new_location(name);                                  +
+    313  struct location *current = realm->castle;                                        +
+    318  struct location *previous = NULL;                                                +
+    319  struct enemy *currentEnemy = current->enemies;                                   +
+    322  struct enemy *previousEnemy = NULL;                                        +
+    350  struct location *current = realm->castle;                                  +
+    354  struct location *previous = NULL;                                          +
+    356  int counter = 0;                                                           +
+    366  struct enemy *currentEnemy = current->enemies;                             +
+    367  struct enemy *previousEnemy = current->enemies;                            +
+    436  struct enemy *currentEnemy = current->enemies;                             +
+    474  int counter = 0;                                                           +
+    476  struct location *current = realm->castle;                                  +
+    485  struct enemy *currentEnemy = current->enemies;                             +
+    505  struct enemy *currentEnemy = current->enemies;                             +
+    531  int counterForBuff = 0;                                                    +
+    536  struct location *current = realm->castle;                                  +
+    556  struct enemy *currentEnemy = current->enemies;                             +
+    590  int counterForMatchedTower = 0;                                            +
+    595  struct location *current = realm->castle;                                  +
+    653  struct enemy *previousEnemy = NULL;                                        +
+    693  int counter = 0;                                                           +
+    695  int counterForRange = 0;                                                   +
+    713  int matching = 0;                                                          +
+    753  int counter = 0;                                                           +
+    755  int counterForRange = 0;                                                   +
+    773  int matching = 0;                                                          +
+    818  struct enemy *previousEnemy = current->enemies;                            +
+    819  struct enemy *currentEnemy = current->enemies;                             +
+    820  struct enemy *nextEnemy = current->enemies->next;                          +
+     72  char string[1024];                                                         +
+     74  int counter = 0;                                                           +
+     90  char string3[1024];                                                        +
+    144  char print_text3[10000];                                                   +
+    225  char string[1024];                                                         +
+    245  char string2[1024];                                                        +
+    281  char string[1024];                                                         +
+                                                                                    +
+ ================================================================================== +
^
+ ========================================================================== +
+                                                                            +
+ Great:                                                                     +
+ ------                                                                     +
+   Header comment                                                           +
+   Whitespace                                                               +
+   Variable names                                                           +
+   Constants (#defines)                                                     +
+   Functions                                                                +
+   Comments                                                                 +
+   Line length                                                              +
+                                                                            +
+                                                                            +
+ A few issues:                                                              +
+ -------------                                                              +
+   Nesting depth (see summary above)                                        +
+                                                                            +
+                                                                            +
+ Poor:                                                                      +
+ -----                                                                      +
+   Indentation (code may not compile - please verify)                       +
+                                                                            +
+ ========================================================================== +
```

Assessment

```
Test stage_one_00_no_loc (./castle_defense) - passed
Test stage_one_00_no_loc_multi (./castle_defense) - passed
Test stage_one_01_one_loc (./castle_defense) - passed
Test stage_one_01_one_loc_multi (./castle_defense) - passed
Test stage_one_02_two_loc (./castle_defense) - passed
Test stage_one_02_two_loc_multi (./castle_defense) - passed
Test stage_one_03_many_loc (./castle_defense) - passed
Test stage_one_03_many_loc_multi (./castle_defense) - passed
Test stage_one_04_extreme_name_loc (./castle_defense) - passed
Test stage_one_04_extreme_name_loc_multi (./castle_defense) - passed
Test stage_two_new_enemy_big_pow_lair (./castle_defense) - passed
Test stage_two_new_enemy_big_pow_land (./castle_defense) - passed
Test stage_two_new_enemy_invalid_powlair (./castle_defense) - passed
Test stage_two_new_enemy_invalid_powtower (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_many_enemy_all_at_castle (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_many_enemy_all_at_lair (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_many_enemy_all_at_land (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_many_enemy_oneeach (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_many_enemy_twoeach (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_one_enemy_all_at_castle (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_one_enemy_all_at_lair (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_one_enemy_all_at_land (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_one_enemy_oneeach (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_one_enemy_twoeach (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_two_enemy_all_at_castle (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_two_enemy_all_at_lair (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_two_enemy_all_at_land (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_two_enemy_oneeach (./castle_defense) - passed
Test stage_two_new_enemy_many_loc_two_enemy_twoeach (./castle_defense) - passed
Test stage_two_new_enemy_no_loc_many_enemy_all_at_castle (./castle_defense) - passed
Test stage_two_new_enemy_no_loc_many_enemy_all_at_lair (./castle_defense) - passed
Test stage_two_new_enemy_no_loc_one_enemy_all_at_castle (./castle_defense) - passed
Test stage_two_new_enemy_no_loc_one_enemy_all_at_lair (./castle_defense) - passed
Test stage_two_new_enemy_no_loc_two_enemy_all_at_castle (./castle_defense) - passed
Test stage_two_new_enemy_no_loc_two_enemy_all_at_lair (./castle_defense) - passed
Test stage_two_new_enemy_two_loc_many_enemy_all_at_castle (./castle_defense) - passed
Test stage_two_new_enemy_two_loc_many_enemy_all_at_lair (./castle_defense) - passed
Test stage_two_new_enemy_two_loc_many_enemy_all_at_land (./castle_defense) - passed
Test stage_two_new_enemy_two_loc_one_enemy_all_at_castle (./castle_defense) - passed
Test stage_two_new_enemy_two_loc_one_enemy_all_at_lair (./castle_defense) - passed
Test stage_two_new_enemy_two_loc_one_enemy_all_at_land (./castle_defense) - passed
Test stage_two_new_enemy_two_loc_two_enemy_all_at_castle (./castle_defense) - passed
Test stage_two_new_enemy_two_loc_two_enemy_all_at_lair (./castle_defense) - passed
Test stage_two_new_enemy_two_loc_two_enemy_all_at_land (./castle_defense) - passed
Test stage_two_new_tower_many_loc_many_tow_after_loc (./castle_defense) - passed
Test stage_two_new_tower_many_loc_many_tow_after_tow (./castle_defense) - passed
Test stage_two_new_tower_many_loc_many_tow_alternate (./castle_defense) - passed
Test stage_two_new_tower_many_loc_many_tow_first (./castle_defense) - passed
Test stage_two_new_tower_many_loc_one_tow_after_loc (./castle_defense) - passed
Test stage_two_new_tower_many_loc_one_tow_after_tow (./castle_defense) - passed
Test stage_two_new_tower_many_loc_one_tow_first (./castle_defense) - passed
Test stage_two_new_tower_many_loc_two_tow_after_loc (./castle_defense) - passed
Test stage_two_new_tower_many_loc_two_tow_after_tow (./castle_defense) - passed
Test stage_two_new_tower_many_loc_two_tow_first (./castle_defense) - passed
Test stage_two_new_tower_no_loc_many_tow_after_tow (./castle_defense) - passed
Test stage_two_new_tower_no_loc_many_tow_first (./castle_defense) - passed
Test stage_two_new_tower_no_loc_one_tow_after_tow (./castle_defense) - passed
Test stage_two_new_tower_invalid_1 (./castle_defense) - passed
Test stage_two_new_tower_invalid_2 (./castle_defense) - passed
Test stage_two_new_tower_no_loc_one_tow_first (./castle_defense) - passed
Test stage_two_new_tower_no_loc_two_tow_after_tow (./castle_defense) - passed
Test stage_two_new_tower_no_loc_two_tow_first (./castle_defense) - passed
Test stage_two_new_tower_one_loc_many_tow_after_loc (./castle_defense) - passed
Test stage_two_new_tower_one_loc_many_tow_after_tow (./castle_defense) - passed
Test stage_two_new_tower_one_loc_many_tow_first (./castle_defense) - passed
Test stage_two_new_tower_one_loc_one_tow_after_loc (./castle_defense) - passed
Test stage_two_new_tower_one_loc_one_tow_after_tow (./castle_defense) - passed
Test stage_two_new_tower_one_loc_one_tow_first (./castle_defense) - passed
Test stage_two_new_tower_one_loc_two_tow_after_loc (./castle_defense) - passed
Test stage_two_new_tower_one_loc_two_tow_after_tow (./castle_defense) - passed
Test stage_two_new_tower_one_loc_two_tow_first (./castle_defense) - passed
Test stage_two_new_tower_two_loc_many_tow_after_loc (./castle_defense) - passed
Test stage_two_new_tower_two_loc_many_tow_after_tow (./castle_defense) - passed
Test stage_two_new_tower_two_loc_many_tow_first (./castle_defense) - passed
Test stage_two_new_tower_two_loc_one_tow_after_loc (./castle_defense) - passed
Test stage_two_new_tower_two_loc_one_tow_after_tow (./castle_defense) - passed
Test stage_two_new_tower_two_loc_one_tow_first (./castle_defense) - passed
Test stage_two_new_tower_two_loc_two_tow_after_loc (./castle_defense) - passed
Test stage_two_new_tower_two_loc_two_tow_after_tow (./castle_defense) - passed
Test stage_two_new_tower_two_loc_two_tow_alternate (./castle_defense) - passed
Test stage_two_new_tower_two_loc_two_tow_first (./castle_defense) - passed
Test stage_two_new_tower_enemy_many_tow_many_enemy_all_at_one (./castle_defense) - passed
Test stage_two_new_tower_enemy_many_tow_many_enemy_all_at_tower (./castle_defense) - passed
Test stage_two_new_tower_enemy_many_tow_many_enemy_twoeach (./castle_defense) - passed
Test stage_two_new_tower_enemy_many_tow_one_enemy_all_at_one (./castle_defense) - passed
Test stage_two_new_tower_enemy_many_tow_one_enemy_all_at_tower (./castle_defense) - passed
Test stage_two_new_tower_enemy_many_tow_one_enemy_twoeach (./castle_defense) - passed
Test stage_two_new_tower_enemy_many_tow_two_enemy_all_at_one (./castle_defense) - passed
Test stage_two_new_tower_enemy_many_tow_two_enemy_all_at_tower (./castle_defense) - passed
Test stage_two_new_tower_enemy_many_tow_two_enemy_twoeach (./castle_defense) - passed
Test stage_two_new_tower_enemy_multi_tow_many_enemy_all_at_one (./castle_defense) - passed
Test stage_two_new_tower_enemy_multi_tow_many_enemy_all_at_tower (./castle_defense) - passed
Test stage_two_new_tower_enemy_multi_tow_many_enemy_twoeach (./castle_defense) - passed
Test stage_two_new_tower_enemy_multi_tow_one_enemy_all_at_one (./castle_defense) - passed
Test stage_two_new_tower_enemy_multi_tow_one_enemy_all_at_tower (./castle_defense) - passed
Test stage_two_new_tower_enemy_multi_tow_one_enemy_twoeach (./castle_defense) - passed
Test stage_two_new_tower_enemy_multi_tow_two_enemy_all_at_one (./castle_defense) - passed
Test stage_two_new_tower_enemy_multi_tow_two_enemy_all_at_tower (./castle_defense) - passed
Test stage_two_new_tower_enemy_multi_tow_two_enemy_twoeach (./castle_defense) - passed
Test stage_two_new_tower_enemy_one_tow_many_enemy_all_at_one (./castle_defense) - passed
Test stage_two_new_tower_enemy_one_tow_many_enemy_all_at_tower (./castle_defense) - passed
Test stage_two_new_tower_enemy_one_tow_one_enemy_all_at_one (./castle_defense) - passed
Test stage_two_new_tower_enemy_one_tow_one_enemy_all_at_tower (./castle_defense) - passed
Test stage_two_new_tower_enemy_one_tow_two_enemy_all_at_one (./castle_defense) - passed
Test stage_two_new_tower_enemy_one_tow_two_enemy_all_at_tower (./castle_defense) - passed
Test stage_two_new_tower_enemy_two_tow_many_enemy_all_at_one (./castle_defense) - passed
Test stage_two_new_tower_enemy_two_tow_many_enemy_all_at_tower (./castle_defense) - passed
Test stage_two_new_tower_enemy_two_tow_one_enemy_all_at_one (./castle_defense) - passed
Test stage_two_new_tower_enemy_two_tow_one_enemy_all_at_tower (./castle_defense) - passed
Test stage_two_new_tower_enemy_two_tow_two_enemy_all_at_one (./castle_defense) - passed
Test stage_two_new_tower_enemy_two_tow_two_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_advance_enemy_many_tow_many_enemy_all_at_one (./castle_defense) - passed
Test stage_three_advance_enemy_many_tow_many_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_advance_enemy_many_tow_many_enemy_twoeach (./castle_defense) - passed
Test stage_three_advance_enemy_many_tow_one_enemy_all_at_one (./castle_defense) - passed
Test stage_three_advance_enemy_many_tow_one_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_advance_enemy_many_tow_one_enemy_twoeach (./castle_defense) - passed
Test stage_three_advance_enemy_many_tow_two_enemy_all_at_one (./castle_defense) - passed
Test stage_three_advance_enemy_many_tow_two_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_advance_enemy_many_tow_two_enemy_twoeach (./castle_defense) - passed
Test stage_three_advance_enemy_multi_tow_many_enemy_all_at_one (./castle_defense) - passed
Test stage_three_advance_enemy_multi_tow_many_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_advance_enemy_multi_tow_many_enemy_twoeach (./castle_defense) - passed
Test stage_three_advance_enemy_multi_tow_one_enemy_all_at_one (./castle_defense) - passed
Test stage_three_advance_enemy_multi_tow_one_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_advance_enemy_multi_tow_one_enemy_twoeach (./castle_defense) - passed
Test stage_three_advance_enemy_multi_tow_two_enemy_all_at_one (./castle_defense) - passed
Test stage_three_advance_enemy_multi_tow_two_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_advance_enemy_multi_tow_two_enemy_twoeach (./castle_defense) - passed
Test stage_three_advance_enemy_one_tow_many_enemy_all_at_one (./castle_defense) - passed
Test stage_three_advance_enemy_one_tow_many_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_advance_enemy_one_tow_one_enemy_all_at_one (./castle_defense) - passed
Test stage_three_advance_enemy_one_tow_one_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_advance_enemy_one_tow_two_enemy_all_at_one (./castle_defense) - passed
Test stage_three_advance_enemy_one_tow_two_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_advance_enemy_two_tow_many_enemy_all_at_one (./castle_defense) - passed
Test stage_three_advance_enemy_two_tow_many_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_advance_enemy_two_tow_one_enemy_all_at_one (./castle_defense) - passed
Test stage_three_advance_enemy_two_tow_one_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_advance_enemy_two_tow_two_enemy_all_at_one (./castle_defense) - passed
Test stage_three_advance_enemy_two_tow_two_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_damage_many_tow_many_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_damage_many_tow_many_enemy_twoeach (./castle_defense) - passed
Test stage_three_damage_many_tow_one_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_damage_many_tow_one_enemy_twoeach (./castle_defense) - passed
Test stage_three_damage_many_tow_two_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_damage_many_tow_two_enemy_twoeach (./castle_defense) - passed
Test stage_three_damage_one_tow_many_enemy_all_at_tower (./castle_defense) - failed (Incorrect output)
Test stage_three_damage_one_tow_one_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_damage_one_tow_two_enemy_all_at_tower (./castle_defense) - failed (Incorrect output)
Test stage_three_damage_two_tow_many_enemy_all_at_tower (./castle_defense) - failed (Incorrect output)
Test stage_three_damage_two_tow_one_enemy_all_at_tower (./castle_defense) - passed
Test stage_three_damage_two_tow_two_enemy_all_at_tower (./castle_defense) - passed
Test stage_four_h_IUAjJCVeMQ___neg6_h_locations (./castle_defense) - passed
Test stage_four_h_U2luZ2xlU2VhcmNo_neg1_h_lair (./castle_defense) - passed
Test stage_four_h_U2luZ2xlU2VhcmNo_neg6_h_lair (./castle_defense) - passed
Test stage_four_h_U2luZ2xlU2VhcmNo_5_h_lair (./castle_defense) - passed
Test stage_four_h_W2EtbV15W24tcF0__neg6_h_lair (./castle_defense) - passed
Test stage_four_h_W2EtbV15_neg6_h_lair (./castle_defense) - passed
Test stage_four_h_W2EtbV1bci16XVtuLXBd_neg6_h_locations (./castle_defense) - passed
Test stage_four_h_YWFhW2Etel1bbS16XVtuLXpd_neg6_h_lair (./castle_defense) - passed
Test stage_four_h_YWFhW3otYV0__neg6_h_locations (./castle_defense) - failed (Incorrect output)
Test stage_four_h_eHlbYS16XQ___neg6_h_locations (./castle_defense) - passed
Test stage_four_h_eltOLVhdYQ___neg1_h_locations (./castle_defense) - passed
Test stage_four_h_eltOLVhdYQ___neg6_h_locations (./castle_defense) - passed
Test stage_four_h_eltOLVhdYQ___5_h_locations (./castle_defense) - passed
Test stage_four_p_U2luZ2xlU2VhcmNo_neg1_p_towers (./castle_defense) - passed
Test stage_four_p_U2luZ2xlU2VhcmNo_neg6_p_towers (./castle_defense) - failed (Incorrect output)
Test stage_four_p_U2luZ2xlU2VhcmNo_5_p_towers (./castle_defense) - passed
Test stage_four_p_W2EtbV15W24tcF0__neg6_p_towers (./castle_defense) - failed (Incorrect output)
Test stage_four_p_W2EtbV15_neg6_p_towers (./castle_defense) - failed (Incorrect output)
Test stage_four_p_YWFhW2Etel1bbS16XVtuLXpd_neg6_p_towers (./castle_defense) - failed (Incorrect output)
Test stage_four_u_IUAjJCVeMQ___neg6_u_towers (./castle_defense) - failed (Incorrect output)
Test stage_four_u_W2EtbV1bci16XVtuLXBd_neg6_u_towers (./castle_defense) - failed (Incorrect output)
Test stage_four_u_eHlbYS16XQ___neg6_u_towers (./castle_defense) - failed (Incorrect output)
Test stage_four_u_eltOLVhdYQ___neg1_u_towers (./castle_defense) - passed
Test stage_four_u_eltOLVhdYQ___neg6_u_towers (./castle_defense) - failed (Incorrect output)
Test stage_four_u_eltOLVhdYQ___5_u_towers (./castle_defense) - passed
Test stage_five_both_1_both (./castle_defense) - failed (Incorrect output)
Test stage_five_both_2_both (./castle_defense) - failed (Incorrect output)
Test stage_five_ice_0_one_ice_tower (./castle_defense) - failed (Incorrect output)
Test stage_five_ice_1_simple_ice_tower (./castle_defense) - passed
Test stage_five_ice_2_ice_tower (./castle_defense) - failed (Incorrect output)
Test stage_five_ice_3_complex_ice_tower (./castle_defense) - failed (Incorrect output)
Test stage_five_portal_0_one_portal_tower (./castle_defense) - failed (Incorrect output)
Test stage_five_portal_1_two_portal_tower (./castle_defense) - passed
Test stage_five_portal_2_simple_portal_towers (./castle_defense) - failed (Incorrect output)
Test stage_five_portal_3_portal_towers (./castle_defense) - failed (Incorrect output)
Test stage_one_01_one_loc_leakcheck (./castle_defense_leakcheck) - passed
Test stage_one_01_one_loc_multi_leakcheck (./castle_defense_leakcheck) - passed
Test stage_one_02_two_loc_leakcheck (./castle_defense_leakcheck) - passed
Test stage_two_new_enemy_two_loc_one_enemy_all_at_land_leakcheck (./castle_defense_leakcheck) - passed
Test stage_two_new_enemy_two_loc_two_enemy_all_at_castle_leakcheck (./castle_defense_leakcheck) - passed
Test stage_two_new_tower_no_loc_many_tow_first_leakcheck (./castle_defense_leakcheck) - passed
Test stage_two_new_tower_no_loc_one_tow_after_tow_leakcheck (./castle_defense_leakcheck) - passed
Test stage_three_advance_enemy_one_tow_one_enemy_all_at_one_leakcheck (./castle_defense_leakcheck) - failed
(errors)
Test stage_three_advance_enemy_one_tow_one_enemy_all_at_tower_leakcheck (./castle_defense_leakcheck) - failed
(errors - same as Test stage_three_advance_enemy_one_tow_one_enemy_all_at_one_leakcheck)
Test stage_three_advance_enemy_two_tow_two_enemy_all_at_tower_leakcheck (./castle_defense_leakcheck) - failed
(errors - same as Test stage_three_advance_enemy_one_tow_one_enemy_all_at_one_leakcheck)
Test stage_three_damage_many_tow_many_enemy_all_at_tower_leakcheck (./castle_defense_leakcheck) - passed
Test stage_three_damage_many_tow_many_enemy_twoeach_leakcheck (./castle_defense_leakcheck) - passed
Test stage_three_damage_many_tow_one_enemy_all_at_tower_leakcheck (./castle_defense_leakcheck) - passed
Test stage_four_h_IUAjJCVeMQ___neg6_h_locations_leakcheck (./castle_defense_leakcheck) - passed
Test stage_four_h_U2luZ2xlU2VhcmNo_neg1_h_lair_leakcheck (./castle_defense_leakcheck) - passed
Test stage_four_h_U2luZ2xlU2VhcmNo_neg6_h_lair_leakcheck (./castle_defense_leakcheck) - passed
Test stage_four_h_U2luZ2xlU2VhcmNo_5_h_lair_leakcheck (./castle_defense_leakcheck) - passed
Test stage_four_h_W2EtbV15W24tcF0__neg6_h_lair_leakcheck (./castle_defense_leakcheck) - passed
Test stage_four_p_W2EtbV15W24tcF0__neg6_p_towers_leakcheck (./castle_defense_leakcheck) - failed (Incorrect output
- same as Test stage_four_p_W2EtbV15W24tcF0__neg6_p_towers)
Test stage_four_p_W2EtbV15_neg6_p_towers_leakcheck (./castle_defense_leakcheck) - failed (Incorrect output - same
as Test stage_four_p_W2EtbV15_neg6_p_towers)
Test stage_four_p_YWFhW2Etel1bbS16XVtuLXpd_neg6_p_towers_leakcheck (./castle_defense_leakcheck) - failed (Incorrect
output - same as Test stage_four_p_YWFhW2Etel1bbS16XVtuLXpd_neg6_p_towers)
Test stage_four_u_IUAjJCVeMQ___neg6_u_towers_leakcheck (./castle_defense_leakcheck) - failed (Incorrect output -
same as Test stage_four_u_IUAjJCVeMQ___neg6_u_towers)
Test stage_four_u_W2EtbV1bci16XVtuLXBd_neg6_u_towers_leakcheck (./castle_defense_leakcheck) - failed (Incorrect
output - same as Test stage_four_u_W2EtbV1bci16XVtuLXBd_neg6_u_towers)
Test stage_four_u_eHlbYS16XQ___neg6_u_towers_leakcheck (./castle_defense_leakcheck) - failed (Incorrect output -
same as Test stage_four_u_eHlbYS16XQ___neg6_u_towers)
Test stage_four_u_eltOLVhdYQ___neg1_u_towers_leakcheck (./castle_defense_leakcheck) - passed
Test stage_four_u_eltOLVhdYQ___neg6_u_towers_leakcheck (./castle_defense_leakcheck) - failed (Incorrect output -
same as Test stage_four_u_eltOLVhdYQ___neg6_u_towers)
Test stage_four_u_eltOLVhdYQ___5_u_towers_leakcheck (./castle_defense_leakcheck) - passed
Test stage_five_both_1_both_leakcheck (./castle_defense_leakcheck) - failed (Incorrect output - same as Test
stage_five_both_1_both)
Test stage_five_both_2_both_leakcheck (./castle_defense_leakcheck) - failed (Incorrect output - same as Test
stage_five_both_2_both)
Test stage_five_ice_0_one_ice_tower_leakcheck (./castle_defense_leakcheck) - failed (errors)
Test stage_five_ice_1_simple_ice_tower_leakcheck (./castle_defense_leakcheck) - failed (errors - same as Test
stage_three_advance_enemy_one_tow_one_enemy_all_at_one_leakcheck)
Test stage_five_ice_2_ice_tower_leakcheck (./castle_defense_leakcheck) - failed (errors)
Test stage_five_ice_3_complex_ice_tower_leakcheck (./castle_defense_leakcheck) - failed (Incorrect output - same as
Test stage_five_ice_3_complex_ice_tower)
Test stage_five_portal_0_one_portal_tower_leakcheck (./castle_defense_leakcheck) - failed (Incorrect output - same
as Test stage_five_portal_0_one_portal_tower)
Test stage_five_portal_1_two_portal_tower_leakcheck (./castle_defense_leakcheck) - passed
Test stage_five_portal_2_simple_portal_towers_leakcheck (./castle_defense_leakcheck) - failed (Incorrect output -
same as Test stage_five_portal_2_simple_portal_towers)
Test stage_five_portal_3_portal_towers_leakcheck (./castle_defense_leakcheck) - failed (Incorrect output - same as
Test stage_five_portal_3_portal_towers)

Marking Summary:

|-------------------------------------------|--------------|--------------|
| Test Name                                 | Tests Passed | % Gained     |
|-------------------------------------------|--------------|--------------|
| [1] Adding & Printing Locations           |  10/10       | 50.0/50      |
| [2] New Enemies                           |  32/32       | 4.0/4        |
| [2] New Enemies (Invalid)                 |  2/2         | 1.0/1        |
| [2] New Towers                            |  35/35       | 4.0/4        |
| [2] New Towers (Invalid)                  |  2/2         | 1.0/1        |
| [2] New Towers & Enemies                  |  30/30       | 5.0/5        |
| [3] Advance Enemies                       |  30/30       | 5.0/5        |
| [3] Apply Damage                          | *9/12        | 4.0/5        |
| [3] Stage 1 Leakcheck                     |  3/3         | 1.0/1        |
| [3] Stage 2 Leakcheck                     |  4/4         | 1.0/1        |
| [3] Stage 3 Leakcheck                     | *3/6         | 1.9/3        |
| [4] Buff Enemy Health                     | *12/13       | 3.8/4        |
| [4] Buff Tower Power                      | *2/6         | 1.6/3        |
| [4] Buff Tower Uses                       | *2/6         | 1.6/3        |
| [No Marks] Stage 4 Leakcheck              |  7/14        | 0.0/0        |
| [5] Ice Effect                            | *1/4         | 2.0/4        |
| [5] Portal Effect                         | *1/4         | 2.0/4        |
| [5] Combined Effects                      |  0/2         | 0.0/2        |
| [No Marks] Stage 5 Leakcheck              |  1/10        | 0.0/0        |

(* indicates test passed some, but not all tests)

NOTE: The following mark may be slightly different to the sum of
the "Mark Allocation" column above. This is due to rounding, not
a mistake. We will not change marks because of rounding.

Sum of percentage points gained: 89.0/100

This mark gets scaled to be out of 70:

Mark for automarking tests: 62.3/70


You can rerun the tests used in marking by running: 1511 automark castle_defense

!!specialmark  (automated testing)                      62.3/70

!!marktab       **  MARKER'S  ASSESSMENT  **
```

```
                            style ..  ..  (20)  16???
 ^
 + ================================================ +
 + Well written code - where you did use functions  +
 + they were used well. Make sure you are           +
 + consistent with camelCase or snake_case and      +
 + consistent with your functions though. There     +
 + were too many cases of overdeep nesting where a  +
 + helper function would have saved you. Also       +
 + avoid commenting the obvious. I feel like half   +
 + of your comments could have been removed as      +
 + they were just pointing out the obvious.         +
 + Otherwise great job and great testing.           +
 + ================================================ +

                            stage1_tests  (4)   4.0???

                            stage2_tests  (3)   3.0???

                            stage3_tests  (2)   2.0???

                            stage4_tests  (1)   1.0???

!!finalmark     **  FINAL  ASSIGNMENT  MARK:    88.3/100

5206267 Luo, Zheng                              3707/3 AEROAH


Marked by z5214808 on Mon Aug 24 20:12:00 2020
```