

# 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

```

1 // Assignment 2 20T2 COMP1511: Castle Defense
2 //
3 // This program was written by Zheng Luo (z5206267@ad.unsw.edu.au)
4 // on July/2020
5 //
6 // Version 1.0.0 (2020-07-20): Assignment released.
    ^
    + ===== +
    + Give a rundown of your approaches here +
    + ===== +

7
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11
12 #include "realm.h"
13 //////////////////////////////////////
14
15 // `struct realm` represents a realm, which represents the state of the
16 // entire program. It is mainly used to point to a linked list of
17 // locations, though you may want to add other fields to it.
18 struct realm {
19     struct location *castle;
20     struct location *lair;
21 };
22
23 // `struct location` represents a location, which could be a land,
24 // a tower, or a castle. Each of those places has a different type
25 // of information that this struct will need to store.
26 struct location {
27     char name[MAX_NAME_LENGTH];
28     int power;
29     int uses;
30     int defense;
31     struct enemy *enemies;
32     struct location *next;
33     Effect effect;
34 };
35
36 // `struct enemy` represents an enemy, which will move through the
37 // realm (to each location). Towers can deal damage to it, and
38 // it can deal damage to the Castle.
39 struct enemy {
40     char name[MAX_NAME_LENGTH];
41     int hp;
42     int hp_Given;
    ^
    + ===== +
    + Make sure you stick to camelCase or snake_case +
    + ===== +
43     struct enemy *next;
44 };
45
46
47 //////////////////////////////////////
48
49 // Function prototypes for helper functions
50 static Location new_location(char *name);
51 static void print_tower(char *name, int power, int uses, Effect effect);
52 static void print_land(char *name);
53 static void print_castle(char *name, int defense);
54 static void print_enemy(char *name, int cur_hp, int max_hp);
55 // Self-created functions below,
56 // its detailed introductions are located in Self-Created functions
57 // after stage 5 function.
58 static void remove_enemy(struct location *current, struct enemy *currentEnemy);
59 static int search_function_tower (struct location *current, char *search_term);
60 static int search_function_enemy (struct enemy *current, char *search_term);
61 static void sorting(struct location *current);
62
63 //////////////////////////////////////

```

```

64
65
66 // Create a new realm, and return a pointer to it.
67 // You may need to change this function in later stages.
68 Realm new_realm(void) {
69     struct realm *realm = malloc(sizeof(struct realm));
70     realm->castle = new_location("Castle");
71     realm->lair = new_location("Lair");
72     // Castle's defense health.
73     realm->castle->defense = STARTING_CASTLE_HP;
74     // Let lair pointing to NULL.
75     realm->lair->next = NULL;
76     // Let the enemy in castle and lair pointing to NULL.
77     ^
78     + ===== +
79     + Dont comment the obvious +
80     + ===== +
81     realm->castle->enemies = NULL;
82     realm->lair->enemies = NULL;
83
84     if (realm->castle != NULL && realm->lair != NULL) {
85         realm->castle->next = realm->lair;
86     }
87
88     return realm;
89 }
90
91 // Return a new location created with malloc.
92 static Location new_location(char *name) {
93     // Allocate memory for new locations.
94     struct location *place = malloc(sizeof(struct location));
95     // Initiate the NULL pointer for place->enemy.
96     place->enemies = NULL;
97     // Initiate the variable for place->power and place->uses.
98     place->power = 0;
99     place->uses = 0;
100     place->effect = EFFECT_NONE;
101     // Assign the name into location.
102     strcpy(place->name, name);
103     // Return pointer for new location.
104     return place;
105 }
106
107 // =====
108 //                               Stage 1 Functions                               //
109 // =====
110
111 // Add a new location to the realm, and return the total number of
112 // locations in the realm.
113 int add_location(Realm realm, char *name) {
114     // Assign the new position as newLand.
115     struct location *newLand = new_location(name);
116     // Assign the corresponded name into newLand.
117     strcpy(newLand->name, name);
118     // Assuming 4 buildings.
119     int counter = 4;
120     // If no building in land yet, which means
121     // castle is connecting to the lair.
122     if (realm->castle->next == realm->lair) {
123         // castle(head)->newLand->Lair->NULL
124         newLand->next = realm->lair;
125         realm->castle->next = newLand;
126         counter--;
127         ^
128         + ===== +
129         + Why assume 4 and then subtract? +
130         + ===== +
131     } else { // Land(s) exist between castle and
132         ^
133         + ===== +
134         +
135         +
136         +
137         +
138         +
139         +
140         +
141         +
142         +
143         +
144         +
145         +
146         +
147         +
148         +
149         +
150         +
151         +
152         +
153         +
154         +
155         +
156         +
157         +
158         +
159         +
160         +
161         +
162         +
163         +
164         +
165         +
166         +
167         +
168         +
169         +
170         +
171         +
172         +
173         +
174         +
175         +
176         +
177         +
178         +
179         +
180         +
181         +
182         +
183         +
184         +
185         +
186         +
187         +
188         +
189         +
190         +
191         +
192         +
193         +
194         +
195         +
196         +
197         +
198         +
199         +
200         +
201         +
202         +
203         +
204         +
205         +
206         +
207         +
208         +
209         +
210         +
211         +
212         +
213         +
214         +
215         +
216         +
217         +
218         +
219         +
220         +
221         +
222         +
223         +
224         +
225         +
226         +
227         +
228         +
229         +
230         +
231         +
232         +
233         +
234         +
235         +
236         +
237         +
238         +
239         +
240         +
241         +
242         +
243         +
244         +
245         +
246         +
247         +
248         +
249         +
250         +
251         +
252         +
253         +
254         +
255         +
256         +
257         +
258         +
259         +
260         +
261         +
262         +
263         +
264         +
265         +
266         +
267         +
268         +
269         +
270         +
271         +
272         +
273         +
274         +
275         +
276         +
277         +
278         +
279         +
280         +
281         +
282         +
283         +
284         +
285         +
286         +
287         +
288         +
289         +
290         +
291         +
292         +
293         +
294         +
295         +
296         +
297         +
298         +
299         +
300         +
301         +
302         +
303         +
304         +
305         +
306         +
307         +
308         +
309         +
310         +
311         +
312         +
313         +
314         +
315         +
316         +
317         +
318         +
319         +
320         +
321         +
322         +
323         +
324         +
325         +
326         +
327         +
328         +
329         +
330         +
331         +
332         +
333         +
334         +
335         +
336         +
337         +
338         +
339         +
340         +
341         +
342         +
343         +
344         +
345         +
346         +
347         +
348         +
349         +
350         +
351         +
352         +
353         +
354         +
355         +
356         +
357         +
358         +
359         +
360         +
361         +
362         +
363         +
364         +
365         +
366         +
367         +
368         +
369         +
370         +
371         +
372         +
373         +
374         +
375         +
376         +
377         +
378         +
379         +
380         +
381         +
382         +
383         +
384         +
385         +
386         +
387         +
388         +
389         +
390         +
391         +
392         +
393         +
394         +
395         +
396         +
397         +
398         +
399         +
400         +
401         +
402         +
403         +
404         +
405         +
406         +
407         +
408         +
409         +
410         +
411         +
412         +
413         +
414         +
415         +
416         +
417         +
418         +
419         +
420         +
421         +
422         +
423         +
424         +
425         +
426         +
427         +
428         +
429         +
430         +
431         +
432         +
433         +
434         +
435         +
436         +
437         +
438         +
439         +
440         +
441         +
442         +
443         +
444         +
445         +
446         +
447         +
448         +
449         +
450         +
451         +
452         +
453         +
454         +
455         +
456         +
457         +
458         +
459         +
460         +
461         +
462         +
463         +
464         +
465         +
466         +
467         +
468         +
469         +
470         +
471         +
472         +
473         +
474         +
475         +
476         +
477         +
478         +
479         +
480         +
481         +
482         +
483         +
484         +
485         +
486         +
487         +
488         +
489         +
490         +
491         +
492         +
493         +
494         +
495         +
496         +
497         +
498         +
499         +
500         +
501         +
502         +
503         +
504         +
505         +
506         +
507         +
508         +
509         +
510         +
511         +
512         +
513         +
514         +
515         +
516         +
517         +
518         +
519         +
520         +
521         +
522         +
523         +
524         +
525         +
526         +
527         +
528         +
529         +
530         +
531         +
532         +
533         +
534         +
535         +
536         +
537         +
538         +
539         +
540         +
541         +
542         +
543         +
544         +
545         +
546         +
547         +
548         +
549         +
550         +
551         +
552         +
553         +
554         +
555         +
556         +
557         +
558         +
559         +
560         +
561         +
562         +
563         +
564         +
565         +
566         +
567         +
568         +
569         +
570         +
571         +
572         +
573         +
574         +
575         +
576         +
577         +
578         +
579         +
580         +
581         +
582         +
583         +
584         +
585         +
586         +
587         +
588         +
589         +
590         +
591         +
592         +
593         +
594         +
595         +
596         +
597         +
598         +
599         +
600         +
601         +
602         +
603         +
604         +
605         +
606         +
607         +
608         +
609         +
610         +
611         +
612         +
613         +
614         +
615         +
616         +
617         +
618         +
619         +
620         +
621         +
622         +
623         +
624         +
625         +
626         +
627         +
628         +
629         +
630         +
631         +
632         +
633         +
634         +
635         +
636         +
637         +
638         +
639         +
640         +
641         +
642         +
643         +
644         +
645         +
646         +
647         +
648         +
649         +
650         +
651         +
652         +
653         +
654         +
655         +
656         +
657         +
658         +
659         +
660         +
661         +
662         +
663         +
664         +
665         +
666         +
667         +
668         +
669         +
670         +
671         +
672         +
673         +
674         +
675         +
676         +
677         +
678         +
679         +
680         +
681         +
682         +
683         +
684         +
685         +
686         +
687         +
688         +
689         +
690         +
691         +
692         +
693         +
694         +
695         +
696         +
697         +
698         +
699         +
700         +
701         +
702         +
703         +
704         +
705         +
706         +
707         +
708         +
709         +
710         +
711         +
712         +
713         +
714         +
715         +
716         +
717         +
718         +
719         +
720         +
721         +
722         +
723         +
724         +
725         +
726         +
727         +
728         +
729         +
730         +
731         +
732         +
733         +
734         +
735         +
736         +
737         +
738         +
739         +
740         +
741         +
742         +
743         +
744         +
745         +
746         +
747         +
748         +
749         +
750         +
751         +
752         +
753         +
754         +
755         +
756         +
757         +
758         +
759         +
760         +
761         +
762         +
763         +
764         +
765         +
766         +
767         +
768         +
769         +
770         +
771         +
772         +
773         +
774         +
775         +
776         +
777         +
778         +
779         +
780         +
781         +
782         +
783         +
784         +
785         +
786         +
787         +
788         +
789         +
790         +
791         +
792         +
793         +
794         +
795         +
796         +
797         +
798         +
799         +
800         +
801         +
802         +
803         +
804         +
805         +
806         +
807         +
808         +
809         +
810         +
811         +
812         +
813         +
814         +
815         +
816         +
817         +
818         +
819         +
820         +
821         +
822         +
823         +
824         +
825         +
826         +
827         +
828         +
829         +
830         +
831         +
832         +
833         +
834         +
835         +
836         +
837         +
838         +
839         +
840         +
841         +
842         +
843         +
844         +
845         +
846         +
847         +
848         +
849         +
850         +
851         +
852         +
853         +
854         +
855         +
856         +
857         +
858         +
859         +
860         +
861         +
862         +
863         +
864         +
865         +
866         +
867         +
868         +
869         +
870         +
871         +
872         +
873         +
874         +
875         +
876         +
877         +
878         +
879         +
880         +
881         +
882         +
883         +
884         +
885         +
886         +
887         +
888         +
889         +
890         +
891         +
892         +
893         +
894         +
895         +
896         +
897         +
898         +
899         +
900         +
901         +
902         +
903         +
904         +
905         +
906         +
907         +
908         +
909         +
910         +
911         +
912         +
913         +
914         +
915         +
916         +
917         +
918         +
919         +
920         +
921         +
922         +
923         +
924         +
925         +
926         +
927         +
928         +
929         +
930         +
931         +
932         +
933         +
934         +
935         +
936         +
937         +
938         +
939         +
940         +
941         +
942         +
943         +
944         +
945         +
946         +
947         +
948         +
949         +
950         +
951         +
952         +
953         +
954         +
955         +
956         +
957         +
958         +
959         +
960         +
961         +
962         +
963         +
964         +
965         +
966         +
967         +
968         +
969         +
970         +
971         +
972         +
973         +
974         +
975         +
976         +
977         +
978         +
979         +
980         +
981         +
982         +
983         +
984         +
985         +
986         +
987         +
988         +
989         +
990         +
991         +
992         +
993         +
994         +
995         +
996         +
997         +
998         +
999         +
1000        +

```

```

+ INLINE COMMENTS: it's usually better for comments to go on the +
+ line above +
+ ===== +
125 // (head)->original    Lair->NULL
126 //          |          |
127 //          ->newLand->
128 // Loop thro until the insert position is reached.
129
130 // realm->land is same as head.
131 // Here current should point to the first newLand, not NULL.
132 struct location *current = realm->castle->next;
133 while (current->next != realm->lair) {
134     counter++;
135     current = current->next;
136 }
137 newLand->next = current->next;
138 current->next = newLand;
139
140 }
141
142 return counter;
143 }
144
145 // Print out the realm by looping through everything in the realm,
146 // including castle/land/tower/lair and enemies under each location.
147 // The print_realm function take realm as input,
148 // no output due to its void type function.
149 void print_realm(Realm realm) {
150     // Pointer "current" start from first land.
151     struct location *current = realm->castle;
152     // Print out castle, lands and lair.
153     // Loop thro until current reached NULL terminator at the end.
154     while (current != NULL) {
155         // Land will be printed if there is no power or uses.
156         if (strcmp(current->name, "Castle") == 0) {
157             // Print out castle.
158             print_castle(realm->castle->name, realm->castle->defense);
159         } else if (current->power == 0 || current->uses == 0) {
160             print_land(current->name);
161         }
162         else {
163             print_tower(current->name, current->power, current->uses,
164                 current->effect);
165         }
166
167         // If there's enemies under the location, proceed.
168         if (current->enemies != NULL) {
169             // Create a new pointer "currentEnemy" at current enemies,
170             // in order to loop thro enemies under each location.
171             struct enemy *currentEnemy = current->enemies;
172             // Loop thro each enemy under the location
173             while (currentEnemy != NULL) {
174                 // Print out enermies.
175                 print_enemy(currentEnemy->name,
176                     currentEnemy->hp, currentEnemy->hp_Given);
177                 // Move to the next enemy under the same location.
178                 currentEnemy = currentEnemy->next;
179             }
180
181         }
182         current = current->next;
183     }
184 }
185
186
187 }
188
189 ////////////////////////////////////////////////////
190 //                               Stage 2 Functions                               //
191 ////////////////////////////////////////////////////
192
193 // Add an enemy to the realm by looping through the list, and insert it

```

```

194 // at the end of the list.
195 // The function take realm, location name, the name of enemy, its hp as input.
196 // The function will return SUCCESS(0) if new enemy if successfully created.
197 int new_enemy(Realm realm, char *location_name, char *name, int hp) {
198     // Ensure that the stats you have been given for HP are not below 1.
199     // If they are, return ERROR_INVALID_STAT.
200     if (hp < 1) {
201         return ERROR_INVALID_STAT;
202     }
203     // Find the Location called location_name.
204     // If one does not exist, return ERROR_NO_LOCATION
205     // Create a pointer current, used to loop thro the list,
206     // in order to match the same name.
207     struct location *current = realm->castle;
208
209     while (current != NULL) {
210         if (strcmp(current->name, location_name) == 0) {
211             ^
212             + ===== +
213             + Could this go in a function find_location()? +
214             + ===== +
215
216             // Place a new enemy, with the given stats,
217             // directly after the last enemy at the location you found above.
218             struct enemy *newEnemy = malloc(sizeof(struct enemy));
219             // Create a pointer "currentEnemy",
220             // in order to insert the new enemy.
221             struct enemy *currentEnemy = NULL;
222             // Assign the name and hp to the enemy.
223             strcpy(newEnemy->name, name);
224             newEnemy->hp = hp;
225             newEnemy->hp_Given = hp;
226             // Link another end of newEnemy to NULL.
227             newEnemy->next = NULL;
228
229             // If there is nothing in the enemies section yet.
230             if (current->enemies == NULL) {
231                 // Link the newEnemy under the tower.
232                 current->enemies = newEnemy;
233             } else { // These is existing enemy in enemies section.
234                 ^
235                 + ===== +
236                 + INLINE COMMENTS: it's usually better for comments to go on the +
237                 + line above +
238                 + ===== +
239
240                 // Loop thro enemies until current->enemies->next == NULL.
241                 currentEnemy = current->enemies;
242                 while (currentEnemy->next != NULL) {
243                     currentEnemy = currentEnemy->next;
244                     ^
245                     + ===== +
246                     + OVERDEEP_NESTING: nesting too deep: 5 (try moving some of the +
247                     + logic into another function) +
248                     + ===== +
249
250                 }
251                 // Inserting newEnemy into list.
252                 currentEnemy->next = newEnemy;
253
254             }
255             // Break the program, so it will not continue to scan the same name
256             // for the rest of the linked list.
257             break;
258             ^
259             + ===== +
260             + Avoid using break +
261             + ===== +
262
263         }
264         current = current->next;
265         if (current == NULL) {
266             return ERROR_NO_LOCATION;
267         }
268     }
269     return SUCCESS;
270 }

```

```

248 }
249
250 // Add a new tower to the realm by looping through the list, and insert it
251 // after previous position.
252 // The function take realm, previous position, the new name of tower,
253 // its power and uses as input.
254 // The function will return SUCCESS(0) if new tower if successfully created.
255 int new_tower(Realm realm, char *prev_name, char *name, int power, int uses) {
256     // Ensure that the stats you have been given for power
257     // and uses are not below 1. If they are, return ERROR_INVALID_STAT.
258
259     ^
260     + ===== +
261     + Avoid the self explanatory +
262     + ===== +
263
264     if (power < 1 || uses < 1) {
265         return ERROR_INVALID_STAT;
266     }
267
268     // Create a pointer current, used to loop thro the list,
269     // in order to match the same name.
270     struct location *current = realm->castle;
271     while (current != NULL) {
272         // If current location is same name as required.
273         if (strcmp(prev_name, current->name) == 0) {
274
275             ^
276             + ===== +
277             + Could this go in a function find_location()? +
278             + ===== +
279
280             // Place a new tower, with the given stats,
281             // directly after the location which you found above.
282             // This tower is inserted into the linked list,
283             // which adds an element and doesn't replace the Location prev_name.
284
285             // Create a new location for new tower.
286             struct location *newTower = new_location(name);
287             // Assign name, power and uses to new tower.
288             strcpy(newTower->name, name);
289             newTower->power = power;
290             newTower->uses = uses;
291             // For stage 5, effect of EFFECT_NONE added to all new tower.
292             newTower->effect = EFFECT_NONE;
293             // Assign the new tower into desired position.
294             newTower->next = current->next;
295             current->next = newTower;
296             // Break the program, so it will not continue to scan the same name
297             // for the rest of the linked list.
298             break;
299         }
300
301         current = current->next;
302         // If current reaches the NULL after the move, return error.
303         if (current == NULL) {
304             return ERROR_NO_LOCATION;
305         }
306     }
307
308     // Return SUCCESS to indicate success.
309     return SUCCESS;
310 }
311
312 // =====
313 //                               Stage 3 Functions                               //
314 // =====
315
316 // Destroy the realm, and free any associated memory by looping through
317 // all the links including both tower/land/castle/lair and enemies.
318 // The function take realm as input,
319 // there is no output as this is void type function.
320 void destroy_realm(Realm realm) {
321     // Free the enemies first, then locations, and lastly castle and lair.
322     // Create two pointers, one "current" to loop thro the linked list,
323     // and one "prev" to keep track of the previous location.

```

```

312 // another "previous" pointing to the memory will be free.
313 struct location *current = realm->castle;
314
315 // Create two while loop to scan thro each enemy in each location.
316 // Free location as decleared in new_location.
317 while (current != NULL) {
318     struct location *previous = NULL;
319     struct enemy *currentEnemy = current->enemies;
320     // Free enemies as decleared in new_enemy.
321     while (currentEnemy != NULL) {
322         struct enemy *previousEnemy = NULL;
323
324         previousEnemy = currentEnemy;
325         currentEnemy = currentEnemy->next;
326         free(previousEnemy);
327     }
328     previous = current;
329     current = current->next;
330     free(previous);
331 }
332 // At the end, free realm as decleared in Realm new_realm(void).
333 free(realm);
334 }
335
336
337
338 /*
339 Advance enemies towards the castle.
340 Advance_enemies will go through the realm,
341 moving each enemy from their current Location to
342 the Location above in the linked list.
343 The function take realm as input,
344 the number of enemies who pass the castle will be recorded as output.
345 */
346 int advance_enemies(Realm realm) {
347     // Create a pointer "current", started at Castle
348     // (We could started From Lair, but required to inverted the link),
349     // used to loop thro the linked list, and move the enemies forward.
350     struct location *current = realm->castle;
351     // Create a pointer "previous", started at NULL,
352     // previous always move one step slower than current,
353     // in order to reference current to pass enemy forward.
354     struct location *previous = NULL;
355     // Create a counter in record the number of enemies have been removed.
356     int counter = 0;
357
358     // Loop thro linked list until NULL is reached.
359     while (current != NULL) {
360         // Check whether the enemy section is empty under current location.
361         // If empty, current move on.
362         // Otherwise, change the link to previous location,
363         // if current is castle, then previous is NULL.
364         if (current->enemies != NULL) {
365             // Create a new pointer to loop thro until NULL is reached.
366             struct enemy *currentEnemy = current->enemies;
367             struct enemy *previousEnemy = current->enemies;
368             if (strcmp(current->name, "Castle") == 0) {
369                 // Check the number of enemies in the castle.
370                 while (currentEnemy != NULL) {
371                     // Number of enemies pass thro castle.
372                     counter++;
373                     currentEnemy = currentEnemy->next;
374                 }
375                 // Then remove all these enemies from the map.
376                 current->enemies = NULL;
377
378                 }

```



```

379 // For stage 5, if tower with effect of ice has been detected.
380 else if (current->power > 0 && current->uses > 0 &&
381 current->effect == EFFECT_ICE) {
382 // Determine the enemy who has HP less than current's power.
383 while (currentEnemy != NULL) {
384 // If current enemy has hp less than tower's power,
385 // then it will be frozen under this tower.
386 // Hence no further action is required other than move pass.
387 if (currentEnemy->hp < current->power) {
388 // Move pass it.
389 previousEnemy = currentEnemy;
390 // ===== +
391 // OVERDEEP_NESTING: nesting too deep: 6 (try moving some of the +
392 // logic into another function) +
393 // ===== +
394 currentEnemy = currentEnemy->next;
395 }
396 // If the first enemy under the tower has hp > power,
397 // who cannot limited to ice effect.
398 // It will advance to next tower as normal.
399 else if (currentEnemy->hp >= current->power &&
400 currentEnemy == current->enemies)
401 {
402 current->enemies = currentEnemy->next;
403 currentEnemy->next = previous->enemies;
404 previous->enemies = currentEnemy;
405 currentEnemy = current->enemies;
406 }
407 // If enemies are not in the two categories above,
408 // then advance to next tower as normal.
409 else {
410 previousEnemy->next = currentEnemy->next;
411 currentEnemy->next = previous->enemies;
412 previous->enemies = currentEnemy;
413 currentEnemy = previousEnemy->next;
414 }
415 }
416 // Enemies will advance to next tower normally
417 // as no effect is applied.
418 else {
419 while (currentEnemy != NULL) {
420 current->enemies = currentEnemy->next;
421 // ===== +
422 // OVERDEEP_NESTING: nesting too deep: 5 (try moving some of the +
423 // logic into another function) (+ 2 other overdeep_nesting errors) +
424 // ===== +
425 currentEnemy->next = previous->enemies;
426 previous->enemies = currentEnemy;
427 currentEnemy = current->enemies;
428 }
429 }
430 // Move previous to the current position.
431 previous = current;
432 // Then move the current to next position in the linked list.
433 current = current->next;
434 }
435 // Function for portal effect below, and reset the pointers.
436 // Since effect portal only applied when the movement is finished,
437 // hence the portal effect function is created after the advance function
438 // is completed.
439 current = realm->castle;
440 while (current != NULL) {
441 struct enemy *currentEnemy = current->enemies;
442 if (current->power > 0 && current->uses > 0 &&
443 current->effect == EFFECT_PORTAL)
444 {
445 // Move all the enemies under portal effect tower back to lair.

```



```

441         while (currentEnemy != NULL) {
442             current->enemies = currentEnemy->next;
443             currentEnemy->next = realm->lair->enemies;
444             realm->lair->enemies = currentEnemy;
445             currentEnemy = current->enemies;
446             // Convert the portal effect back to none effect,
447             // after the portal effect has been applied once.
448             current->effect = EFFECT_NONE;
449         }
450
451     }
452     // This is sorting function, more detailed see function description.
453     // which sorts enemies alphabetically under the same location.
454     // As all the effects and re-arrangements are completed,
455     // we should sort the enemies alphabetically to conclude the function.
456     if (current->enemies != NULL) {
457         sorting(current);
458     }
459
460     current = current->next;
461 }
462 return counter;
463 }
464
465
466
467 // Apply damage from the enemies at each tower to that tower.
468 // Go through each Enemy at that Tower, and reduce it's HP by the Tower's power.
469 // After the forward movements have been done to enemies,
470 // apply damage to enemies, and reduce use by one to tower.
471 // This function take realm as input, the number of damaged enemies as outputs.
472 int apply_damage(Realm realm) {
473     // Initiate a counter to record the number of enemies damaged this way.
474     int counter = 0;
475     // Create a pointer "current" to loop thro locations from castle till NULL.
476     struct location *current = realm->castle;
477
478     while (current != NULL) {
479         // Proceed if current location is tower.
480         // Tower is when it has both non-zero power and uses.
481         if (current->uses != 0 && current->power != 0 &&
482             current != realm->castle) {
483             // Check the number of enemies in the location.
484             // Create a new pointer to loop thro until NULL is reached.
485             struct enemy *currentEnemy = current->enemies;
486
487             // Apply damage to all enemies under the tower.
488             while (currentEnemy != NULL) {
489                 // Apply damage to current enemy.
490                 currentEnemy->hp = (currentEnemy->hp) - current->power;
491                 // Reduce tower's use by one.
492                 current->uses--;
493                 // The number of damaged enemy increase by one.
494                 counter++;
495                 // Move on.
496                 currentEnemy = currentEnemy->next;
497             }
498             remove_enemy(current, currentEnemy);
499
500         }
501         else if (current == realm->castle) { // When enemy is under castle.

```

```

===== +
+ INLINE COMMENTS: it's usually better for comments to go on
the +
+ line above
+
+
===== +
502     // Enemies can cause castle's defense to decrease.
503     // Check the number of enemies in the castle.
504     // ...

```

```

504         // Create a new pointer to loop thro until NULL is reached.
505         struct enemy *currentEnemy = current->enemies;
506         while (currentEnemy != NULL) {
507             // Castle's remaining health.
508             realm->castle->defense -= currentEnemy->hp;
509             currentEnemy = currentEnemy->next;
510         }
511     }
512     current = current->next;
513 }
514
515 return counter;
516 }
517
518
519 ////////////////////////////////////////////////////
520 //                               Stage 4 Functions                               //
521 ////////////////////////////////////////////////////
522
523 /*
524 The function of "apply_buff" apply the specified buff
525 to the relevant towers or enemies based on search term and buff type.
526 This funtion take realm, search term, buff type, and amount of buff as inputs.
527 The number of buff will be applied are the outputs of this function.
528 */
529 int apply_buff(Realm realm, char *search_term, Buff buff, int amount){
530     // Record number of buff has been applied.
531     int counterForBuff = 0;
532     // Loop through linked list, to determine amount of
533     // towers or enemies are matching the search terms.
534     // Based on the effect type, different function can be achieved.
535     // Create a pointer "current" to matching cases.
536     struct location *current = realm->castle;
537     while (current != NULL) {
538         // Seperate tower and enemies into two different types.
539         if (current->power != 0 && current->uses != 0 &&
540             buff != BUFF_ENEMY_HP &&
541             search_function_tower(current, search_term) == 1) { // Tower
                    ^
                    +
===== +
+ INLINE COMMENTS: it's usually better for
comments to go on the +
+ line above
+
+
===== +
542         // Apply buff.
543         if (buff == BUFF_TOWER_POWER) {
544             // Increase each of their power by the specified amount.
545             current->power += amount;
546             // Return the number of towers you found this way.
547             counterForBuff++;
548         } else if (buff == BUFF_TOWER_USES) {
549             // Increase each of their uses by the specified amount.
550             current->uses += amount;
551             counterForBuff++;
552         }
553
554     } else if (current->enemies != NULL) { // Enemies.
555         // Create a pointer "currentEnemy" to matching cases.
556         struct enemy *currentEnemy = current->enemies;
557         // Loop thro each enemy under the location until NULL is reached.
558         while (currentEnemy != NULL) {
559             // Apply buffs to enemy.
560             if (buff == BUFF_ENEMY_HP &&
561                 search_function_enemy(currentEnemy, search_term) == 1) {
562                 // Increase each of their HP by the specified amount.
563                 currentEnemy->hp += amount;
                    ^
                    + ===== +
+ OVERDEEP_NESTING: nesting too deep: 5 (try moving some of the +

```

```

+ logic into another function) +
+ ===== +
564 // Return the number of enemies you found this way.
565 counterForBuff++;
566 }
567 currentEnemy = currentEnemy->next;
568 }
569 // If debuff has been applied,
570 // enemies' hp has decreased, then remove enemy.
571 remove_enemy(current, currentEnemy);
572 }
573 current = current->next;
574 }
575 return counterForBuff;
576 }
577
578
579 ////////////////////////////////////////////////////
580 //                               Stage 5 Functions                               //
581 ////////////////////////////////////////////////////
582
583 /*
584 Apply the effect described by effect_type to every tower matching search_term.
585 The function take realm, search term, and effect type as inputs,
586 the number of towers that match search term will be returned.
587 */
588 int apply_effect(Realm realm, char *search_term, Effect effect) {
589 // Record number of towers that have been matched.
590 int counterForMatchedTower = 0;
591 // Loop through linked list, to determine amount of
592 // towers are matching the search terms.
593 // Based on the effect type, different function can be achieved.
594 // Create a pointer "current" to matching cases.
595 struct location *current = realm->castle;
596 while (current != NULL) {
597 // Locate the towers matching search term.
598 if (current->power > 0 && current->uses > 0 &&
599 search_function_tower(current, search_term)) {
600 // Then determine different effect.
601 if (effect == EFFECT_ICE)
602 {
603 // If an Enemy would move from the Tower with this effect
604 // to the next Location,
605 // and that enemy has HP less than to the current tower's power,
606 // it stays at the current Tower.
607
608 // Assign effect to the tower.
609 current->effect = EFFECT_ICE;
610 counterForMatchedTower++;
611 }
612 else if (effect == EFFECT_PORTAL)
613 {
614 /*
615 After all enemies have moved,
616 if there are enemies at portal towers,
617 they should all be moved back to the Lair.
618 This effect is removed from a tower after
619 it has finished moving enemies.
620 */
621 // Assign effect to the tower.
622 current->effect = EFFECT_PORTAL;
623 counterForMatchedTower++;
624 }
625 }
626 current = current->next;
627 }
628 return counterForMatchedTower;
629 }
630
631
632 ////////////////////////////////////////////////////
633 //                               Self-Created Functions                               //
634 ////////////////////////////////////////////////////

```

```

634 ///////////////////////////////////////////////////
635
636 /*
637 The function of "remove_enemy" removes the enemy who has hp not above zero.
638 The function takes current location (tower/land), and currentEnemy as inputs.
639 It will scan through every single enemies under the location,
640 and remove it if hp < 0 is detected (free node), then re-do the process,
641 until all enemies have hp > 0.
642 This function is void type, hence no output will be produced.
643 */
644 static void remove_enemy(struct location *current, struct enemy *currentEnemy) {
645     // Remove enemies.
646     // Reset the currentEnemy, loop thro the enemies again,
647     // in order to check any enemies'hp are below 0.
648     currentEnemy = current->enemies;
649     // Create a pointer "previousEnemy",
650     // which is move one step slower than currentEnemy,
651     // in order to link the previousEnemy and currentEnemy->next,
652     // when currentEnemy is removed.
653     struct enemy *previousEnemy = NULL;
654     while (currentEnemy != NULL) {
655         // Check whether hp is below 0.
656         if (currentEnemy->hp <= 0) {
657             // If this is the first enemy under the tower.
658             if (current->enemies == currentEnemy) {
659                 current->enemies = currentEnemy->next;
660                 free(currentEnemy);
661                 // Reset currentEnemy back to head.
662                 currentEnemy = current->enemies;
663             } else { // Else this is not the first enemy under.
664                 ^
665                 + ===== +
666                 + INLINE COMMENTS: it's usually better for comments to go on the +
667                 + line above (+ 1 other inline +
668                 + comments errors) +
669                 + ===== +
670                 previousEnemy->next = currentEnemy->next;
671                 free(currentEnemy);
672                 // Reset.
673                 currentEnemy = current->enemies;
674                 previousEnemy = NULL;
675             }
676         } else { // hp > 0, move on.
677             previousEnemy = currentEnemy;
678             currentEnemy = currentEnemy->next;
679         }
680     }
681 }
682
683 /*
684 Search function for tower and enemy by looping through each elements,
685 and conduct required searches.
686 These search functions are implementing prefix search, not exact match.
687 which means the word can pass the test as long as
688 the search term can match the word for the same length.
689 Both functions take current location (either tower or enemy),
690 and required search term as inputs.
691 They will return 1 if matching, otherwise return 0.
692 These two functions are basically have the same content, but different input,
693 search_function_tower takes struct location type of current as input,
694 search_function_enemy takes struct enemy type of current as input.
695 ^
696 + ===== +
697 + Great function comment +
698 + ===== +
699
700 /*
701 static int search_function_tower (struct location *current, char *search_term) {
702     // Initiate a counter for counting each word in current.
703     int counter = 0;
704     // Counting each word for search term.
705     int counterForRange = 0;
706     // ...

```

```

696 // Prefix search:
697 // Search will finished if looped thro search term.
698 // Return 0 if any character is not matching,
699 // return 1 only if all search terms are exactly matching.
700 while (counterForRange < strlen(search_term)) {
701
702     // If no [] is detected, match each character.
703     if (search_term[counterForRange] != '[' ) {
704         if (search_term[counterForRange] == current->name[counter]) {
705             // matched, move on to next character.
706             ^
707             + ===== +
708             + Avoid empty conditions - invert the logic +
709             + instead +
710             + ===== +
711         } else {
712             // Not matched, end the function.
713             return 0;
714         }
715     } else if (search_term[counterForRange] == '[') {
716         // Each character will have a unique matching number,
717         // If it is still 1 at the end of the
718         int matching = 0;
719         // Then we need to find the location of ].
720         while (search_term[counterForRange] != ']') {
721             // For single matching stype input. e.g.[abc]
722             if (search_term[counterForRange] == current->name[counter]) {
723                 // Then matched.
724                 matching = 1;
725                 ^
726                 + ===== +
727                 + OVERDEEP_NESTING: nesting too deep: 5 (try moving some of the +
728                 + logic into another function) +
729                 + ===== +
730             }
731             // For range style input e.g.[a-z]
732             else if (search_term[counterForRange] == '-') {
733                 if (search_term[counterForRange - 1] <=
734                     current->name[counter]
735                     && search_term[counterForRange + 1] >=
736                     current->name[counter]) {
737                     // Matched
738                     matching = 1;
739                     ^
740                     + ===== +
741                     + OVERDEEP_NESTING: nesting too deep: 6 (try moving some of the +
742                     + logic into another function) +
743                     + ===== +
744                 }
745             }
746             counterForRange++;
747         }
748         // If the programme didnt found corresponded characters,
749         // at the end of ]
750         // Which means no matched.
751         if (matching != 1) {
752             return 0;
753         }
754     }
755     counter++;
756     counterForRange++;
757 }
758 return 1;
759 }
760
761 // Search function for enemy.
762 // Return 1 if completely matched.
763 // Detailed description is above.
764 static int search_function_enemy (struct enemy *current, char *search_term) {
765     // Initiate a counter for counting each word in current.
766     ^

```

```

753     int counter = 0;
754     // Counting each word for search term.
755     int counterForRange = 0;
756     // Prefix search:
757     // Search will finished if looped thro search term.
758     // Return 0 if any character is not matching,
759     // return 1 only if all search terms are exactly matching.
760     while (counterForRange < strlen(search_term)) {
761
762         // If no [] is detected, match each character.
763         if (search_term[counterForRange] != '[' ) {
764             if (search_term[counterForRange] == current->name[counter]) {
765                 // matched, move on to next character.
766             } else {
767                 // Not matched, end the function.
768                 return 0;
769             }
770         } else if (search_term[counterForRange] == '[') {
771             // Each character will have a unique matching number,
772             // If it is still 1 at the end of the
773             int matching = 0;
774             // Then we need to find the location of ].
775             while (search_term[counterForRange] != ']') {
776                 // For single matching stype input. e.g.[abc]
777                 if (search_term[counterForRange] == current->name[counter]) {
778                     // Then matched.
779                     matching = 1;
780                     ^
781                     + ===== +
782                     + OVERDEEP_NESTING: nesting too deep: 5 (try moving some of the +
783                     + logic into another function) +
784                     + ===== +
785                 }
786                 // For range style input e.g.[a-z]
787                 else if (search_term[counterForRange] == '-') {
788                     if (search_term[counterForRange - 1] <=
789                         current->name[counter]
790                         && search_term[counterForRange + 1] >=
791                         current->name[counter]) {
792                         // Matched
793                         matching = 1;
794                         ^
795                         + ===== +
796                         + OVERDEEP_NESTING: nesting too deep: 6 (try moving some of the +
797                         + logic into another function) +
798                         + ===== +
799                     }
800                 }
801                 counterForRange++;
802             }
803             // If the programme didnt found corresponded characters,
804             // at the end of ]
805             // Which means no matched.
806             if (matching != 1) {
807                 return 0;
808             }
809             counter++;
810             counterForRange++;
811         }
812     }
813     return 1;
814 }
815
816 /*
817 The function "sorting" sorts all enemies alphabetically based on their names
818 under the same location. This function is achieved similar to the
819 bubble sort technique, which compares two variables, then swap, move to the
820 next position, and then repeat until all variables are in alphabetical order.
821 The function take struct location * type as input, and no output based on its

```



```

815 void type.
816 */
817 static void sorting(struct location *current) {
818     struct enemy *previousEnemy = current->enemies;
819     struct enemy *currentEnemy = current->enemies;
820     struct enemy *nextEnemy = current->enemies->next;
821     while (nextEnemy != NULL) {
822         while (strcmp(currentEnemy->name, nextEnemy->name) > 0) {
823             // Z > A.
824             // First number
825             if (currentEnemy == current->enemies) {
826                 currentEnemy->next = nextEnemy->next;
827                 nextEnemy->next = current->enemies;
828                 current->enemies = nextEnemy;
829                 // reset
830                 previousEnemy = current->enemies;
831                 currentEnemy = current->enemies;
832                 nextEnemy = current->enemies->next;
833             }
834             else {/// Assuming in the middle of role.
835                 currentEnemy->next = nextEnemy->next;
836                 previousEnemy->next = nextEnemy;
837                 nextEnemy->next = currentEnemy;
838                 // reset
839                 previousEnemy = current->enemies;
840                 currentEnemy = current->enemies;
841                 nextEnemy = current->enemies->next;
842             }
843         }
844     }
845     previousEnemy = currentEnemy;
846     currentEnemy = nextEnemy;
847     nextEnemy = nextEnemy->next;
848 }
849
850
851 }
852
853
854
855
856 ////////////////////////////////////////
857 //                               Provided print functions                               //
858 //               NOTE: YOU SHOULD NOT MODIFY THE FOLLOWING FUNCTIONS                //
859 ////////////////////////////////////////
860
861 void print_tower(char *name, int power, int uses, int effect) {
862     printf(" ^ %32s [pow: %3d | uses: %3d]", name, power, uses);
863     if (effect == EFFECT_NONE) printf(" {%c}", EFFECT_NONE_CHAR);
864     if (effect == EFFECT_PORTAL) printf(" {%c}", EFFECT_PORTAL_CHAR);
865     if (effect == EFFECT_ICE) printf(" {%c}", EFFECT_ICE_CHAR);
866     printf("\n");
867 }
868
869 void print_land(char *name) {
870     printf(" ^ %32s [_____]\n", name);
871 }
872
873 void print_castle(char *name, int defense) {
874     printf(" ^ %32s [Castle Defenses: %3d]\n", name, defense);
875 }
876
877 void print_enemy(char *name, int cur_hp, int max_hp) {
878     printf(" ^ %40s [hp: %d/%d]\n", name, cur_hp, max_hp);
879 }
880
881 ////////////////////////////////////////
882 //                               End of provided print functions                               //
883 ////////////////////////////////////////

```



887

```

1 // Assignment 2 20T2 COMP1511: Castle Defense
2 //
3 // This program was written by Zheng Luo (z5206267@ad.unsw.edu.au)
4 // on July/2020
5 //
6 // Version 1.0.0 (2020-07-20): Assignment released.
7 // Version 1.0.1 (2020-07-21): Add return value to main.
8
9 #include <stdio.h>
10 #include <string.h>
11 #include <assert.h>
12
13 #include "realm.h"
14 #include "test_realm.h"
15 #include "capture.h"
16
17 ////////////////////////////////////////////////////
18 //                               Function Prototypes                               //
19 ////////////////////////////////////////////////////
20
21 // TODO: Add your own prototypes here, if necessary.
22
23 ////////////////////////////////////////////////////
24 //                               Helper Functions                               //
25 ////////////////////////////////////////////////////
26
27 // Find the string 'needle' in 'haystack'
28 int string_contains(char *haystack, char *needle) {
29     return strstr(haystack, needle) != NULL;
30 }
31
32 // NOTE: You should not declare any structs from realm.c here.
33 // Declaring structs from realm.c in this function is against the
34 // rules of the assignment.
35
36 // Main function: this function will not be used in testing, you do not
37 // need to change it.
38 // If you want to write your own tests, add them in `extra_tests`.
39 int main(int argc, char *argv[]) {
40     printf("\n===== Castle Defense Tests =====\n");
41
42     test_add_location();
43     test_print_realm();
44     test_new_enemy();
45     test_new_tower();
46     test_apply_damage();
47     extra_tests();
48
49     return 0;
50 }
51
52 // NOTE: These tests are explained in `test_realm.h`
53
54 void test_add_location(void) {
55     printf("Test whether this add_location follows the spec: ");
56
57     // Test 1: Does add_location's return value count the Castle & Lair?
58
59     Realm test_realm = new_realm();
60
61     int num_locations = add_location(test_realm, "Location");
62     if (num_locations != 3) {
63         printf(DOES_NOT_FOLLOW_SPEC);
64         // since we don't want to print FOLLOWS_SPEC at the end,
65         // we just return to the main function here.
66         return;
67     }
68
69     // Test 2: Does add_location return the correct amount for lots of locations?
70     // Test if 1024 locations will be added.
71     Realm test_realm_test2 = new_realm();
72     // ...

```

```

72     char string[1024];
73     num_locations = 0;
74     int counter = 0;
75     while (counter < 1024) {
76         // store difference names into string based on counter.
77         sprintf(string, "Location%d", counter);
78         // the name of location will be based on string.
79         num_locations = add_location(test_realm_test2, string);
80         counter++;
81     }
82     if (num_locations != 1026) {
83         printf(DOES_NOT_FOLLOW_SPEC);
84         return;
85     }
86
87     // Test 3: Add your own test, and explain it below:
88     // Test whether added location is located before lair.
89     Realm test_realm_test3 = new_realm();
90     char string3[1024];
91     char expected_string[1024] =
92     " ^                               Castle [Castle Defenses: 100]\n\
93     ^                               Location1 [_____] \n\
94     ^                               Location2 [_____] \n\
95     ^                               Lair [_____] \n";
96     // Two locations will be added in test 3,
97     add_location(test_realm_test3, "Location1");
98     add_location(test_realm_test3, "Location2");
99     // the location2 must located before lair and after location1.
100    // castle->location1->location2->lair
101    CAPTURE(print_realm(test_realm_test3), string3, 1024);
102    if (strcmp(string3, expected_string) != 0) {
103        printf(DOES_NOT_FOLLOW_SPEC);
104        return;
105    }
106
107
108    printf(FOLLOWS_SPEC);
109 }
110
111 void test_print_realm(void) {
112     printf("Test whether this print_realm follows the spec: ");
113
114     // Test 1: Does print_realm show the Castle?
115
116     Realm test_realm = new_realm();
117
118     char print_text[10000];
119     CAPTURE(print_realm(test_realm), print_text, 10000);
120
121     if (!string_contains(print_text, "Castle")) {
122         printf(DOES_NOT_FOLLOW_SPEC);
123         // since we don't want to print FOLLOWS_SPEC at the end,
124         // we just return to the main function here.
125         return;
126     }
127
128     // Test 2: Does print_realm show the correct HP of the castle?
129
130     Realm test2_realm = new_realm();
131
132     char print_text_test2[10000];
133     CAPTURE(print_realm(test2_realm), print_text_test2, 10000);
134
135     if (!string_contains(print_text_test2, "Castle Defenses: 100")) {
136         printf(DOES_NOT_FOLLOW_SPEC);
137         return;
138     }
139
140     // Test 1: Does print_realm show the Lair?
141
142     Realm test3_realm = new_realm();
143
144     char print_text_test3[10000];

```

```

144     char print_text3[10000];
145     CAPTURE(print_realm(test3_realm), print_text3, 10000);
146
147     if (!string_contains(print_text, "Lair")) {
148         printf(DOES_NOT_FOLLOW_SPEC);
149         return;
150     }
151
152     printf(FOLLOWS_SPEC);
153 }
154
155 // Stage 2
156
157 void test_new_enemy(void) {
158     printf("Test whether this new_enemy follows the spec: ");
159
160     // Test 1: Does new_enemy work if you're adding to the Lair?
161     Realm test_realm = new_realm();
162     // Generate a tested enemy call "Roger" with 5hp at Lair.
163     // This is following the spec, if return value is SUCCESS.
164     if (new_enemy(test_realm, "Lair" , "Roger", 5) != SUCCESS) {
165         printf(DOES_NOT_FOLLOW_SPEC);
166         return;
167     }
168
169     // Test 2: Does new_enemy work if you're adding to the Castle?
170     Realm test2_realm = new_realm();
171     // Generate a tested enemy call "Roger" with 5hp at Castle.
172     // This is following the spec, if return value is SUCCESS.
173     if (new_enemy(test2_realm, "Castle" , "Roger", 5) != SUCCESS) {
174         printf(DOES_NOT_FOLLOW_SPEC);
175         return;
176     }
177     printf(FOLLOWS_SPEC);
178 }
179
180 void test_new_tower(void) {
181     printf("Test whether this new_tower follows the spec: ");
182
183     // Test 1: Does new_tower work if you're adding after another tower?
184     Realm test_realm = new_realm();
185     // Create first test tower right after the Castle with power and uses 5.
186     new_tower(test_realm, "Castle", "TestTower1", 5, 5);
187     // Then create second test tower after test tower 1.
188     // If second tower does not return success, then print "NOT FOLLOW SPEC"
189     if (new_tower(test_realm, "TestTower1", "TestTower2", 5, 5) != SUCCESS) {
190         printf(DOES_NOT_FOLLOW_SPEC);
191         return;
192     }
193
194     // Test 2: Does new_tower work if it's power or uses are less than 1?
195     Realm test2_realm = new_realm();
196     // Create a tower with power 0, uses 0.
197     // if it return success, then print "NOT FOLLOW SPEC".
198     if (new_tower(test2_realm, "Castle", "TestTower", 0, 0) == SUCCESS) {
199         printf(DOES_NOT_FOLLOW_SPEC);
200         return;
201     }
202
203     // Test 3: Does new_tower work if there is no previous location
204     // in the link as given in the function?
205     Realm test3_realm = new_realm();
206     // Create a tower after a non-existing location.
207     // If it return success, then print "NOT FOLLOW SPEC".
208     if (new_tower(test3_realm, "NON_EXISTING_LOCATION",
209         "TestTower", 5, 5) == SUCCESS) {
210         printf(DOES_NOT_FOLLOW_SPEC);
211         return;
212     }
213
214     printf(FOLLOWS_SPEC);
215 }

```

```

216
217 // Stage 3 (2 marks)
218
219 void test_apply_damage(void) {
220
221     printf("Test whether this apply_damage follows the spec: ");
222
223     // Test 1: Does apply_damage actually destroy enemies?
224     Realm test_realm = new_realm();
225     char string[1024];
226
227
228     // Create a tower and a enemy under the tower,
229     // with a hp lower than tower's power.
230     new_tower(test_realm, "Castle", "TestTower", 5, 5);
231     new_enemy(test_realm, "TestTower", "TestEnemy", 3);
232     // Apply damage
233     apply_damage(test_realm);
234     CAPTURE(print_realm(test_realm), string, 1024);
235
236     if (string_contains(string, "TestEnemy")) {
237         printf(DOES_NOT_FOLLOW_SPEC);
238         return;
239     }
240
241
242     // Test 2: Could apply damage function destory and damage multiple enemies
243     // under the same location at the same time?
244     Realm test2_realm = new_realm();
245     char string2[1024];
246     // Create a tower and multiple enemies under the tower, with different hp.
247     new_tower(test2_realm, "Castle", "TestTower", 5, 5);
248     new_enemy(test2_realm, "TestTower", "TestEnemy1", 3);
249     new_enemy(test2_realm, "TestTower", "TestEnemy2", 10);
250     new_enemy(test2_realm, "TestTower", "TestEnemy3", 3);
251     // Apply damage
252     apply_damage(test2_realm);
253     CAPTURE(print_realm(test2_realm), string2, 1024);
254     if (string_contains(string2, "TestEnemy1") ||
255         string_contains(string2, "TestEnemy3")) {
256         printf(DOES_NOT_FOLLOW_SPEC);
257         return;
258     }
259
260     printf(FOLLOWS_SPEC);
261 }
262
263 // Stage 4 (1 marks)
264
265 void test_apply_buff(void) {
266     printf("Test whether this apply_buff follows the spec: ");
267     // Test 1: Does the buff has been successfully added?
268     // Apply 5 power buff to TestTower, if the function is not returning 1
269     // then print DOES NOT FOLLOW SPEC.
270     Realm test_realm = new_realm();
271     new_tower(test_realm, "Castle", "TestTower", 5, 5);
272     if (apply_buff(test_realm, "TestTower", BUFF_TOWER_POWER, 5) != 1) {
273         printf(DOES_NOT_FOLLOW_SPEC);
274         return;
275     }
276
277     // Test 2: Does the buff has been added correctly?
278     // Apply 5 power buff to TestTower with 5 power originally,
279     // if it is not returning 10 as buff, then return DOES NOT FOLLOW SPEC.
280     Realm test2_realm = new_realm();
281     char string[1024];
282     new_tower(test2_realm, "Castle", "TestTower", 5, 5);
283     apply_buff(test2_realm, "TestTower", BUFF_TOWER_POWER, 5);
284     CAPTURE(print_realm(test2_realm), string, 1024);
285     if (!string_contains(string, "TestTower [pow: 10 | uses: 5] {n}")) {
286         printf(DOES_NOT_FOLLOW_SPEC);
287         return;
288     }
289 }

```

```
288     }
289
290     printf(FOLLOWS_SPEC);
291 }
292
293     ^
294     + ===== +
295     + NOTE: broken indentation starts here -- watch out for stray +
296     + curly brackets, or functions with ';' not '{' +
297     + ===== +
298 // Extra Tests (not worth marks, but fun!)
299
300 void extra_tests(void) {
301     // TODO: add extra tests, if you'd like to.
302 }
```

## 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 int i;
+ 323 int j;
+ 324 int k;
+ 325 int l;
+ 326 int m;
+ 327 int n;
+ 328 int o;
+ 329 int p;
+ 330 int q;
+ 331 int r;
+ 332 int s;
+ 333 int t;
+ 334 int u;
+ 335 int v;
+ 336 int w;
+ 337 int x;
+ 338 int y;
+ 339 int z;
+ 340 int a;
+ 341 int b;
+ 342 int c;
+ 343 int d;
+ 344 int e;
+ 345 int f;
+ 346 int g;
+ 347 int h;
+ 348 int i;
+ 349 int j;
+ 350 int k;
+ 351 int l;
+ 352 int m;
+ 353 int n;
+ 354 int o;
+ 355 int p;
+ 356 int q;
+ 357 int r;
+ 358 int s;
+ 359 int t;
+ 360 int u;
+ 361 int v;
+ 362 int w;
+ 363 int x;
+ 364 int y;
+ 365 int z;
+ 366 int a;
+ 367 int b;
+ 368 int c;
+ 369 int d;
+ 370 int e;
+ 371 int f;
+ 372 int g;
+ 373 int h;
+ 374 int i;
+ 375 int j;
+ 376 int k;
+ 377 int l;
+ 378 int m;
+ 379 int n;
+ 380 int o;
+ 381 int p;
+ 382 int q;
+ 383 int r;
+ 384 int s;
+ 385 int t;
+ 386 int u;
+ 387 int v;
+ 388 int w;
+ 389 int x;
+ 390 int y;
+ 391 int z;
+ 392 int a;
+ 393 int b;
+ 394 int c;
+ 395 int d;
+ 396 int e;
+ 397 int f;
+ 398 int g;
+ 399 int h;
+ 400 int i;
+ 401 int j;
+ 402 int k;
+ 403 int l;
+ 404 int m;
+ 405 int n;
+ 406 int o;
+ 407 int p;
+ 408 int q;
+ 409 int r;
+ 410 int s;
+ 411 int t;
+ 412 int u;
+ 413 int v;
+ 414 int w;
+ 415 int x;
+ 416 int y;
+ 417 int z;
+ 418 int a;
+ 419 int b;
+ 420 int c;
+ 421 int d;
+ 422 int e;
+ 423 int f;
+ 424 int g;
+ 425 int h;
+ 426 int i;
+ 427 int j;
+ 428 int k;
+ 429 int l;
+ 430 int m;
+ 431 int n;
+ 432 int o;
+ 433 int p;
+ 434 int q;
+ 435 int r;
+ 436 int s;
+ 437 int t;
+ 438 int u;
+ 439 int v;
+ 440 int w;
+ 441 int x;
+ 442 int y;
+ 443 int z;
+ 444 int a;
+ 445 int b;
+ 446 int c;
+ 447 int d;
+ 448 int e;
+ 449 int f;
+ 450 int g;
+ 451 int h;
+ 452 int i;
+ 453 int j;
+ 454 int k;
+ 455 int l;
+ 456 int m;
+ 457 int n;
+ 458 int o;
+ 459 int p;
+ 460 int q;
+ 461 int r;
+ 462 int s;
+ 463 int t;
+ 464 int u;
+ 465 int v;
+ 466 int w;
+ 467 int x;
+ 468 int y;
+ 469 int z;
+ 470 int a;
+ 471 int b;
+ 472 int c;
+ 473 int d;
+ 474 int e;
+ 475 int f;
+ 476 int g;
+ 477 int h;
+ 478 int i;
+ 479 int j;
+ 480 int k;
+ 481 int l;
+ 482 int m;
+ 483 int n;
+ 484 int o;
+ 485 int p;
+ 486 int q;
+ 487 int r;
+ 488 int s;
+ 489 int t;
+ 490 int u;
+ 491 int v;
+ 492 int w;
+ 493 int x;
+ 494 int y;
+ 495 int z;
+ 496 int a;
+ 497 int b;
+ 498 int c;
+ 499 int d;
+ 500 int e;
+ 501 int f;
+ 502 int g;
+ 503 int h;
+ 504 int i;
+ 505 int j;
+ 506 int k;
+ 507 int l;
+ 508 int m;
+ 509 int n;
+ 510 int o;
+ 511 int p;
+ 512 int q;
+ 513 int r;
+ 514 int s;
+ 515 int t;
+ 516 int u;
+ 517 int v;
+ 518 int w;
+ 519 int x;
+ 520 int y;
+ 521 int z;
+ 522 int a;
+ 523 int b;
+ 524 int c;
+ 525 int d;
+ 526 int e;
+ 527 int f;
+ 528 int g;
+ 529 int h;
+ 530 int i;
+ 531 int j;
+ 532 int k;
+ 533 int l;
+ 534 int m;
+ 535 int n;
+ 536 int o;
+ 537 int p;
+ 538 int q;
+ 539 int r;
+ 540 int s;
+ 541 int t;
+ 542 int u;
+ 543 int v;
+ 544 int w;
+ 545 int x;
+ 546 int y;
+ 547 int z;
+ 548 int a;
+ 549 int b;
+ 550 int c;
+ 551 int d;
+ 552 int e;
+ 553 int f;
+ 554 int g;
+ 555 int h;
+ 556 int i;
+ 557 int j;
+ 558 int k;
+ 559 int l;
+ 560 int m;
+ 561 int n;
+ 562 int o;
+ 563 int p;
+ 564 int q;
+ 565 int r;
+ 566 int s;
+ 567 int t;
+ 568 int u;
+ 569 int v;
+ 570 int w;
+ 571 int x;
+ 572 int y;
+ 573 int z;
+ 574 int a;
+ 575 int b;
+ 576 int c;
+ 577 int d;
+ 578 int e;
+ 579 int f;
+ 580 int g;
+ 581 int h;
+ 582 int i;
+ 583 int j;
+ 584 int k;
+ 585 int l;
+ 586 int m;
+ 587 int n;
+ 588 int o;
+ 589 int p;
+ 590 int q;
+ 591 int r;
+ 592 int s;
+ 593 int t;
+ 594 int u;
+ 595 int v;
+ 596 int w;
+ 597 int x;
+ 598 int y;
+ 599 int z;
+ 600 int a;
+ 601 int b;
+ 602 int c;
+ 603 int d;
+ 604 int e;
+ 605 int f;
+ 606 int g;
+ 607 int h;
+ 608 int i;
+ 609 int j;
+ 610 int k;
+ 611 int l;
+ 612 int m;
+ 613 int n;
+ 614 int o;
+ 615 int p;
+ 616 int q;
+ 617 int r;
+ 618 int s;
+ 619 int t;
+ 620 int u;
+ 621 int v;
+ 622 int w;
+ 623 int x;
+ 624 int y;
+ 625 int z;
+ 626 int a;
+ 627 int b;
+ 628 int c;
+ 629 int d;
+ 630 int e;
+ 631 int f;
+ 632 int g;
+ 633 int h;
+ 634 int i;
+ 635 int j;
+ 636 int k;
+ 637 int l;
+ 638 int m;
+ 639 int n;
+ 640 int o;
+ 641 int p;
+ 642 int q;
+ 643 int r;
+ 644 int s;
+ 645 int t;
+ 646 int u;
+ 647 int v;
+ 648 int w;
+ 649 int x;
+ 650 int y;
+ 651 int z;
+ 652 int a;
+ 653 int b;
+ 654 int c;
+ 655 int d;
+ 656 int e;
+ 657 int f;
+ 658 int g;
+ 659 int h;
+ 660 int i;
+
```



```

+ 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

[illegible]

[illegible]



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\_elt0LVhdYQ\_\_neg1\_h\_locations (./castle\_defense) - passed  
Test stage\_four\_h\_elt0LVhdYQ\_\_neg6\_h\_locations (./castle\_defense) - passed  
Test stage\_four\_h\_elt0LVhdYQ\_\_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\_elt0LVhdYQ\_\_neg1\_u\_towers (./castle\_defense) - passed  
Test stage\_four\_u\_elt0LVhdYQ\_\_neg6\_u\_towers (./castle\_defense) - failed (Incorrect output)  
Test stage\_four\_u\_elt0LVhdYQ\_\_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_elt0LVhdYQ___neg1_u_towers_leakcheck (./castle_defense_leakcheck) - passed
Test stage_four_u_elt0LVhdYQ___neg6_u_towers_leakcheck (./castle_defense_leakcheck) - failed (Incorrect output -
same as Test stage_four_u_elt0LVhdYQ___neg6_u_towers)
Test stage_four_u_elt0LVhdYQ___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
```

**COMP1511 20T2: Programming Fundamentals** is brought to you by  
the [School of Computer Science and Engineering](#)  
at the [University of New South Wales](#), Sydney.  
For all enquiries, please email the class account at [cs1511@cse.unsw.edu.au](mailto:cs1511@cse.unsw.edu.au)  
CRICOS Provider 00098G