# Recitation Class

## Week 7

# Arrangements for Future Quizzes and Homeworks

- **Project 1**
  - **Project 1 has been released, the deadline is Nov 4, 20:00 P.M. 2025**
  - **Project 1 requires to make a Plants vs. Zombies! Enjoy it! (30 pts)**
  - **And there is also a code assignment about Hash Table, which yields 20 pts.**
  - **Please Submit your Project 1 via Gradescope and remember to match each question correctly.**
- **Quiz**
  - **Quiz 2 will be held during the recitation class today. Remember to bring a pencil or pen!**
  - **It covers the contents from slides DS-1 to DS-5(Array, Linked List, Stack, Queue, Hash Table, Asymptotic Analysis)**
  - **In the future, quiz will be held after the content of each chapter is finished.**
- **Check:**
  - **We will check your project 1 next week. Grading policy is attached to the .zip file.**
  - **In the future, checks will only be held for projects, to ensure your code is done by yourself.**

# Homework

A

### 3. Queue-as-array with dynamic resizing

A queue uses an array `Q[N]` with initial $N = 8$ and the *one-slot-empty* rule:

$$\texttt{empty} \iff \texttt{front} = \texttt{rear}, \qquad \texttt{full} \iff (\texttt{rear} + 1) \bmod N = \texttt{front}.$$

**Resize:** at any time when the queue is full, allocate size $2N$, copy the logical order so that the old front becomes `Q'[0]`, then set `front` $= 0$, `rear` = (old size), $N \leftarrow 2N$, finally place the new element at `Q'[rear]` and advance `rear`.

Initially:

$$\texttt{front} = 6, \quad \texttt{rear} = 1, \quad N = 8,$$

and from front to `rear` $- 1$ the content is

$$A\ (Q[6]),\ B\ (Q[7]),\ C\ (Q[0]).$$

Do in order (assume no underflow):

$$\texttt{push}(D),\ \texttt{push}(E),\ \texttt{pop}(),\ \texttt{push}(F),\ \texttt{push}(G),\ \texttt{push}(H),\ \texttt{push}(I),\ \texttt{pop}(),\ \texttt{push}(J),\ \texttt{push}(K).$$

What is the final state (from front to `rear` $- 1$) and indices?

A. `front` $= 1$, `rear` $= 10$, $N = 16$; content: $C$, $D$, $E$, $F$, $G$, $H$, $I$, $J$, $K$.

B. `front` $= 8$, `rear` $= 3$, $N = 16$; content: $I$, $J$, $K$.

C. `front` $= 7$, `rear` $= 2$, $N = 16$; content: $B$, $C$, $D$, $E$, $F$, $G$, $H$.

D. `front` $= 0$, `rear` $= 9$, $N = 16$; content: $C$, $D$, $E$, $F$, $G$, $H$, $I$, $J$, $K$.

Queue FIFO

Stack LIFO

front 头指针 rear 尾指针

Initially, rear. front

0 1 2 3 4 5 6 7

C D E F G H B full

Resize:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

~~B~~ C D E F G H I J K

front

rear.

# Homework

**5. Which of the following is correct?**

A. $\log n = o\left(n^{0.0001}\right)$.

B. $n \log n = o(n)$.

C. $2^{\sqrt{n}} = o\left(n^{10}\right)$.

D. $n^{\log\log n} = o\left((\log n)^{\log n}\right)$.

E. $n! = o(2^n)$.

$\mathrm{f}(n) = \mathbf{o}(\mathrm{g}(n))$ $\quad \lim_{n\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} = 0$

$\mathrm{f}(n) = \mathbf{O}(\mathrm{g}(n))$ $\quad \lim_{n\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} < \infty$

$\mathrm{f}(n) = \mathbf{\Theta}(\mathrm{g}(n))$ $\quad 0 < \lim_{n\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} < \infty$

$\mathrm{f}(n) = \mathbf{\Omega}(\mathrm{g}(n))$ $\quad \lim_{n\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} > 0$

$\mathrm{f}(n) = \boldsymbol{\omega}(\mathrm{g}(n))$ $\quad \lim_{n\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} = \infty$

A. $\log n = o(n^c)$ $c>0$.

$$\lim_{n\to\infty} \frac{\log n}{n^c} = \lim_{n\to\infty} \frac{\frac{1}{n}}{c n^{c-1}} = \lim_{n\to\infty} \frac{1}{c n^c} = 0$$

$\therefore \log n = o(n^c)$

D. $n^{\log\log n} = (e^{\ln n})^{\ln\ln n} = e^{\ln n \ln\ln n}$

$(\log n)^{\log n} = (e^{\ln(\ln n)})^{\ln n} = e^{\ln\ln n \cdot \ln n} = e^{\ln n \ln\ln n}$

E. $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

# Homework

**(a)** Prove $n^3 = O(n^4)$ using a limit-based argument. Show the key limit and state the conclusion clearly. **(1 pts)**

$\lim_{n\to\infty} \frac{n^3}{n^4} = \lim_{n\to\infty} \frac{1}{n} = 0$ so $n^3 = O(n^4)$

**(b)** Find an $f(n), g(n) \geq 0$ such that $f(n) = O(g(n))$ yet $\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0$. Provide a concrete pair example and a one-line justification. What's the case when $f(n), g(n) \geq 0$ such that $f(n) = O(g(n))$ yet $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ doen't exist? Also provide a concrete pair example and a one-line justification.
**(2 pts)** *Hint: think about oscillating functions!* sinx cosx

1° $f(n)=3n$ $g(n)=5n$

$\therefore \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{3n}{5n} = \frac{3}{5}$

meaning that $f(n) = O(g(n))$

However, it's still true that $f(n) = O(g(n))$

2° $f(x) = x(\sin x + 1)$

$g(x) = x$

$\therefore \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \sin n + 1$ limit doesn't exists.

$\because \sin x + 1 \leq 2 \quad \therefore f(x) \leq 2x = 2 \cdot g(x) \Rightarrow f(n) = O(g(n))$

# Homework

**(c)** Order the functions $f_1, \ldots, f_9$ from smaller to larger asymptotic growth so that if $f_i$ appears before $f_j$, then $f_i = O(f_j)$. Write only the order as a list like $f_8, f_9, \ldots$. No justification is required. **(2 pts)**

Constant

$$f_1(n) = 3^n, \quad f_2(n) = n^{1/3}, \quad f_3(n) = 12, \quad f_4(n) = 2^{\log_2 n},$$
$$f_5(n) = \sqrt{n}, \quad f_6(n) = 2^n, \quad f_7(n) = \log_2 n, \quad f_8(n) = 2^{\sqrt{n}}, \quad f_9(n) = n^3.$$

$= n$

$f_3, f_7, f_2, f_5, f_4, f_9, f_8, f_6, f_1$

**(d)** For each pair, indicate $O$, $\Omega$, or $\Theta$ for $f(n) = ?(g(n))$. Show the key limit and state the conclusion clearly. **(2 pts)**

(i) $f(n) = \log_3 n,\ g(n) = \log_4 n$

(ii) $f(n) = n \log(n^4),\ g(n) = n^2 \log(n^3)$

(iii) $f(n) = \sqrt{n},\ g(n) = (\log n)^3$

(iv) $f(n) = n + \log n,\ g(n) = n + (\log n)^2$

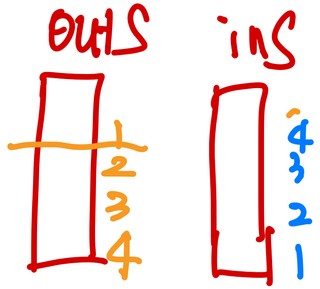
(i) $f = \Theta(g)$
$f = O(g)$ $f = \Omega(g)$

(ii) $f = O(g)$

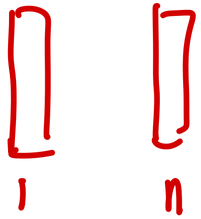
(iii) $f = \Omega(g)$

(iv) $f = \Theta(g)$. $f = O(g)$
$f = \Omega(g)$

```
    std::stack<Type> inS, outS;
    long long pop_count = 0;

    void transfer_if_needed() {
        if (/*(1)*/) {
            while (/*(2)*/) {
                Type v = /*(3)*/;
                ++pop_count;
                outS.push(v);
            }
        }
    }

public:
    bool empty() const { return inS.empty() && outS.empty(); }

    void push(Type const &obj) {
        inS.push(obj);
        transfer_if_needed();
    }

    Type pop() {
        if (empty()) { throw underflow(); }
        Type v = /*(4)*/;
        ++pop_count;
        transfer_if_needed();
        return v;
    }

    long long pops() const { return pop_count; }
};
```




先进后出 → 先进先出.




# Homework

Fill:

(1) outS.empty( ) (2) !inS.empty( )

(3) inS.pop( ) (4) outS.pop( )

**(b) Worst-case and amortized bounds. (2 pts)**

For `push` in `TwoStackQueue`, the worst-case time is $\Theta(1)$ and the amortized time is $\Theta(1)$. For `pop` in `TwoStackQueue`, the worst-case time is $\Theta(n)$ and the amortized time is $\Theta(1)$.

1. `push(1), push(2), push(3), push(4), push(5), push(6), push(7), push(8).`
2. `pop(), pop(), pop().`
3. `push(9), push(10), push(11), push(12).`
4. `pop(), pop(), pop(), pop(), pop().`
5. `push(13), push(14), push(15), push(16).`
6. `pop(), pop(), pop(), pop(), pop(), pop().`

Answer: 30

Last In First Out

First In First Out

# Homework

**1. Stack and Queue. Which of the following statements are correct?** ACD

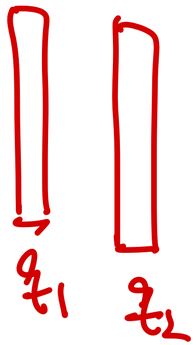
A. A stack (LIFO) can be used to *reverse* the order of a sequence by pushing all items and then popping them.

B. A queue (FIFO) can be used to reverse the order of a sequence by pushing all items and then popping them.

C. Using two stacks, we can implement a queue with `push` and `pop` operations whose *amortized* time is $\Theta(1)$ per operation.

D. Using two queues, we can implement a stack so that each `push` is $\Theta(1)$ worst-case, but `pop` may be $\Theta(n)$ in the worst case.

E. For both stacks and queues, accessing the $k$-th stored element is $\Theta(1)$ time.




push: $x \rightarrow q_1$

pop: 前 $n-1$ 个 $q_1$ 元素 $\rightarrow q_2$，弹 $q_1$ 的最后的元素，swap($q_1$, $q_2$)

# Homework

**2. Hash Table**

Open addressing with *double hashing*, table size $M = 17$. Primary hash $h_1(k) = k \bmod 17$, secondary step $h_2(k) = 1 + (k \bmod 16)$.

**Insertion/search policy:** probe $i_j = (h_1(k) + j \cdot h_2(k)) \bmod M$ for $j = 0, 1, \ldots$. Use *lazy erasing*: bins may be marked `ERASED`; when **searching**, treat `ERASED` as *occupied* and continue and stop when it meets `EMPTY` or the matching key; when **inserting**, treat `ERASED` as *unoccupied* (permitting placement).

After some operations, the table snapshot (index → content) is:

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| content | EMPTY | 18 | EMPTY | EMPTY | EMPTY | ERASED | 52 | ERASED | 86 |
| index | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | |
| content | 103 | 120 | 137 | EMPTY | EMPTY | EMPTY | EMPTY | EMPTY | |

Which of the following are *correct* under this snapshot and policy?

A. Key 137 must have been inserted *after* key 18.

B. It's possible that exactly one erase operation produced the two `ERASED` bins at indices 5 and 7.

C. If we now insert a key $y$ with $h_1(y) = 11$ and $h_2(y) = 1$, then $y$ will be placed at index 12.

D. There exists a key $x \equiv 11 \pmod{17}$ such that `search`$(x)$ inspects exactly 5 bins on this snapshot.

E. It is possible that key 52 was the very first key inserted into an initially empty table.

A. Insert 35
Insert 137
Erase 18
Insert 18

D. $x = 62$

$\therefore h_1(k) = 11 \quad h_2(k) = 15$

$\therefore$ ① 11
② $26 \bmod 17 = 9$
③ $24 \bmod 17 = 7$
④ $22 \bmod 17 = 5$
⑤ $20 \bmod 17 = 3$
⋮

# Homework

**3. Which of the options for $T(n)$ share the same $\Theta$-asymptotic solution?**
Assume $T(0) = T(1) = 1$.

A. $T(n) = 2T(n/2) + \Theta(n)$ $\Theta(n\log n)$
B. $T(n) = T(n-1) + n$ $\Theta(n^2)$
C. $T(n) = 3T(n/3) + n$ $\Theta(n\log n)$
D. $T(n) = T(n/2) + \Theta(n)$ $\Theta(n)$
E. $T(n) = 4T(n/2) + \Theta(n^2)$ $\Theta(n^2\log n)$

A. $T(n) = 2T(\frac{n}{2}) + \Theta(n)$

$= 2\cdot\left(2T(\frac{n}{4}) + \Theta(\frac{n}{2})\right) + \Theta(n)$

$= 4T(\frac{n}{4}) + \Theta(n) + \Theta(n)$

$= 4T(\frac{n}{4}) + 2\Theta(n)$

$2^2$ $= 4\cdot(2T(\frac{n}{8}) + \Theta(\frac{n}{4})) + 2\Theta(n)$

$2^3$ $= 8T(\frac{n}{8}) + 3\Theta(n) = \dots = \Theta(n\log n)$

B. $T(n) = T(n-1) + n$

$= T(n-2) + (n-1) + n$

$= \dots$

$= \frac{(1+n)n}{2} = \Theta(n^2)$

C. $T(n) = 3T(\frac{n}{3}) + n$

$= 9T(\frac{n}{9}) + n + n$

$= \dots$

$= n\cdot\log_3 n$

$= \Theta(n\log n)$

D. $T(n) = T(\frac{n}{2}) + \Theta(n)$

$= T(\frac{n}{4}) + \Theta(\frac{n}{2}) + \Theta(n)$

$= T(\frac{n}{8}) + \Theta(\frac{n}{4}) + \Theta(\frac{n}{2}) + \Theta(n)$

$= \dots$

$= \Theta\left(n\left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right)\right)$

$= \Theta\left(n\cdot\frac{1\,(+(\frac{1}{2})^{\log n})}{1-\frac{1}{2}}\right)$

$= \Theta(n)$

E. $T(n) = 4T(\frac{n}{2}) + \Theta(n^2)$

$= 4\left(4T(\frac{n}{4}) + \Theta((\frac{n}{2})^2)\right) + \Theta(n^2)$

$= 4^2T(\frac{n}{2^2}) + \Theta(n^2) + \Theta(n^2)$

$= 4^{\log_2 n} + \Theta(n^2\log_2 n)$

$= \Theta(n^2\log n)$

# Homework

**4. Asymptotics Analysis**

Assume `op()` runs in constant time. Consider the following two procedures; their running times (as functions of input size $n$) are $F(n)$ and $G(n)$, respectively.

```
void AlgoF(int n){
  op();
  if (n % 2 == 0) {
    int r = ceil(sqrt(n));
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < r; ++j)
        op();
  }
}

void AlgoG(int n){
  int L = floor(log2(max(n,1)));
  for (int i = 0; i < n; ++i)
    for (int t = 0; t < L; ++t)
      op();
}
```

Which of the following statements are true about $F(n)$ and $G(n)$?

A. $F(n) = o(n^2)$. ✓

B. $F(n) = \Omega(n)$. ✗

C. $F(n) + G(n) = \Theta(n^{1.5})$. ✗

D. $F(n) + G(n) = \omega(n \log(1.5n))$.

$f(n) = n\lceil\sqrt{n}\rceil(1+(-1)^n)+1$

$g(n) = n\lfloor\log n\rfloor$

A. $f(n) \le 2n\lceil\sqrt{n}\rceil+1 \sim 2n^{1.5} = O(n^2)$

B. $f(n) = \Omega(g(n))$. $\exists c \in \mathbb{R}^+, \exists n_0, \forall n > n_0, 0 \le g(n) \le c \cdot f(n)$

$f(n) \neq \Omega(n) \Leftrightarrow \neg(\exists c \in \mathbb{R}^+, \exists n_0, \forall n > n_0, 0 \le n \le c \cdot f(n))$

$\Leftrightarrow \forall c \in \mathbb{R}^+, \forall n_0, \exists n > n_0, (n < 0) \vee (n > c \cdot f(n))$

This holds because you can always find such $n$ that $n > c \cdot f(n)$

odd

C. $f(n)+g(n) \neq \Omega(n^{1.5}) \Leftrightarrow \neg(\exists c \in \mathbb{R}^+, \exists n_0, \forall n > n_0. 0 \le n^{1.5} \le c \cdot (f(n)+g(n)))$

$\Leftrightarrow \forall c \in \mathbb{R}^+, \forall n_0, \exists n > n_0, (n^{1.5} < 0) \vee (n^{1.5} > c \cdot (f(n)+g(n)))$

This holds: $n^{1.5} > c \cdot (f(n)+g(n)) = c + c \cdot g(n)$ because $n^{1.5} = \omega(c + cg(n))$

odd $\Theta = \Omega \wedge O \Rightarrow f(n)+g(n) \neq \Theta(n^{1.5})$

D. $f(n)+g(n) \neq \omega(n \log(1.5n)) \Leftrightarrow \neg(\forall c \in \mathbb{R}^+, \exists n_0, \forall n > n_0, 0 \le n\log(1.5n) \le c \cdot (f(n)+g(n)))$

$\Leftrightarrow \exists c \in \mathbb{R}^+, \forall n_0, \exists n > n_0, (n\log(1.5n) < 0) \vee (n\log(1.5n) > c \cdot (f(n)+g(n)))$

holds? let $c = 1$.

$n\log(1.5n) > f(n)+g(n) = 1 + n\lfloor\log n\rfloor$

# Homework

**5. Considering a hash table using open addressing, which of the following statements are correct?**

A. If the table size $m$ is prime. The hash function is $h_1(k) = k \bmod m$, and the step function is $h_2(k) = 1 + (k \bmod (m-1))$. The probe sequence is $i_j = (h_1(k) + j \cdot h_2(k)) \bmod m$ for $j = 0, 1, \ldots, m-1$. Then for any key, the probe sequence will visit every slot, unless it terminates early on `EMPTY`.

B. If the table size $m$ is prime. The hash function is $h_1(k) = k \bmod m$, and the probe sequence is $i_j = (h_1(k) + \frac{1}{2}(j + j^2)) \bmod m$ for $j = 0, 1, \ldots, m-1$. Then for any key, the probe sequence will visit every slot, unless it terminates early on `EMPTY`.

C. In linear probing, as an implementation of erasing an element, moving all successive elements forward one slot until meets an empty slot preserves the correctness of future searches; however, large load factor can degrade performance unless adaptively growth of hash table size is performed.

D. In quadratic probing, as an implementation of erasing an element, using a special `ERASED` marker (distinct from `EMPTY`) preserves the correctness of future searches; however, excessive `ERASED` lables can degrade performance unless periodic rehashing is performed.

A. ∵ m is prime

∴ gcd $(h_2, m) = 1$

∴ if two probes are the same

$(h_1 + jh_2) \equiv (h_1 + j'h_2) \pmod m$

$\Rightarrow (j - j')h_2 \equiv 0 \pmod m$

∵ gcd$(h_2, m) = 1$

∴ $j \equiv j' \pmod m$

∵ $j, j' = 0, 1 \cdots m-1$

∴ $j = j'$

B. $i_j = (h_1 + \frac{1}{2}(j + j^2)) \bmod m$

$a = m - 1 - j \equiv -1 - j \pmod m$

∴ $a^2 + a \equiv (-1-j)^2 + (-1-j) \pmod m$

$= (1 + 2j + j^2) - 1 - j$

$= j^2 + j$

∴ $a^2 + a \equiv j^2 + j \pmod m$

∴ $(m-1-j)^2 + (m-1-j) \equiv j^2 + j \pmod m$

∵ m is prime → it's odd

∴ 2 is invertible under mod m

∴ $\frac{j^2+j}{2} \equiv \frac{(m-1-j)^2 + (m-1-j)}{2} \bmod m.$

# Homework

**Move the two nodes after p to the front, preserving order. (3 pts)**

Complete `move_two_to_first(Node&* head, Node* p)`. You may assume `p->next` and `p->next->next` exist.

```
void move_two_to_first(Node&* head, Node* p) {
  Node* x = /* (1) */;
  Node* y = /* (2) */;
  p->next  = /* (3) */;
  /* (4) */;
  /* (5) */;
}
```

Fill:

(1) p→next

(2) p→next→next

(3) y→next

(4) y→next =head

(5) head=x

# Homework

(e) $T(n) = 3T(n-2) + 5$ **(1 pts)**

$= 3^2 T(n-4) + 5\cdot 3 + 5$

$= 3^3 T(n-6) + 5\cdot 3^2 + 5\cdot 3 + 5$

$= \cdots$

$= 3^{\lfloor \frac{n}{2} \rfloor} T(n \bmod 2) + 5\cdot 3^{\lfloor \frac{n}{2} \rfloor - 1} + 5\cdot 3^{\lfloor \frac{n}{2} \rfloor - 2} + \cdots + 5\cdot 3^2 + 5\cdot 3 + 5$

$= 3^{\lfloor \frac{n}{2} \rfloor} + \frac{5\cdot(3^{\lfloor \frac{n}{2} \rfloor} - 1)}{3-1} = \frac{7}{2}\cdot 3^{\lfloor \frac{n}{2} \rfloor} - \frac{5}{2} = \Theta(3^{\lfloor \frac{n}{2} \rfloor})$

# Homework

**(f)** $T(n) = 3\,T(n^{1/3}) + \Theta(\log n)$ **(1 pts)**

let $x = \log n$

$\therefore T(e^x) = 3T(e^{\frac{x}{3}}) + \Theta(x)$

let $S = T(e^x) = T(n)$

$\therefore S(x) = 3S(\frac{x}{3}) + \Theta(x)$

$\therefore S(x) = 3S(\frac{x}{3}) + \Theta(x) = 9S(\frac{x}{9}) + \Theta(x)$

$+ 3 \cdot \Theta(\frac{x}{3})$

$= 27S(\frac{x}{27}) + 9 \cdot \Theta(\frac{x}{9}) + 3 \cdot \Theta(\frac{x}{3}) + \Theta(x)$

$= \cdots = \Theta(x \log x)$

$\therefore T(n) = S(x) = \Theta(x \log x)$

$= \Theta(\log n \log\log n)$

# Homework

**(bonus)** $T(n) = T(n-1) + T(n-2)$ **(2 pts)**

Squeeze + guess & check method.

1° lowerbound it by $T(n) \geq 2T(n-2)$

so we know $T(n) = \Omega(2^{\frac{n}{2}})$

2° upperbound it by $T(n) \leq 2T(n-1)$

so we know $T(n) = O(2^n)$

Hence, $T(n) = 2^{\Theta(n)}$

More precisely, we know that runtime is exponential w.r.t. n.

$\therefore T(n) = \Theta(2^n)$

$\therefore 2^n = 2^{n-1} + 2^{n-2}$

$\therefore 2 = \frac{1 \pm \sqrt{5}}{2}$

$\because 2$ must be positive

$\therefore 2 = \frac{1+\sqrt{5}}{2}$

$\therefore T(n) = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$

# Project 1

# Project 1


stu
名称
修改日期
大小
种类
1-Project1-概览.docx
昨天 下午 10:48
466 KB
Micros...(.docx)
2-样例游戏参考.docx
10/24/25 下午 8:30
576 KB
Micros...(.docx)
3-建议的开始流程.md
5/27/25 下午 11:06
4 KB
Markdo...ument
4-grading.md
昨天 下午 11:10
8 KB
Markdo...ument
5-FAQ.md
昨天 下午 11:07
27 KB
Markdo...ument
attachment
昨天 下午 10:45
--
文件夹


attachment
名称
修改日期
大小
种类
assets
6/15/25 下午 2:34
--
文件夹
build
6/16/25 下午 1:42
--
文件夹
CMakeLists.txt
5/28/25 下午 4:00
1 KB
纯文本文稿
include
昨天 下午 10:45
--
文件夹
src
6/16/25 下午 1:41
--
文件夹
test.c
6/16/25 下午 2:34
197 字节
C Source
test.exe
6/16/25 下午 2:34
76 KB
Windo...Archive
third_party
6/15/25 下午 8:13
--
文件夹

# Grading Policy

## 基础功能实现 30 pts

- 以下的功能为加分项，功能应该正确实现没有 bug，并且拥有合适的贴图与动画显示。
  - (3 pts) 点击向日葵种子，能成功在场地上种下向日葵并显示，植物种子有冷却时间。
  - (3 pts) 向日葵可以产生阳光，场地中也会随时间规律产生随机掉落阳光，阳光可以被收集用于种植植物。
  - (3 pts) 场地右侧会生成僵尸，僵尸会按照一定速度朝左侧移动，僵尸到达屏幕左侧后游戏失败。
  - (6 pts) 僵尸会啃咬植物造成伤害，植物血量与啃咬伤害数值设置合理，植物血量归零后消失。
  - (3 pts) 可以种植豌豆射手，豌豆射手可以按照一定攻击速度发射豌豆。
  - (6 pts) 射出的豌豆将会对第一个碰到的僵尸造成伤害，碰撞后豌豆将消失。
  - (6 pts) 成功实现铲子功能，当拿起铲子后若第一次点击到的是植物，那么将会移除该植物。
- 需要注意的是，以上条目中实际上包含一些隐式的要求，比如游戏应该能稳定运行、在未点击过植物种子的情况下点击草坪不应该种下植物、种下植物之后再点击草坪应该什么都不会发生、除了向日葵之外其它植物不应该生产阳光（除非你特意实现了杂交功能）、铲子被拿起并铲掉了一个植物之后就应该被自动放下等等。如果你的程序在这些方面有错误，你的得分将是 **unspecified** 的。所以，在实现更多植物和僵尸之前，请务必先确保这些基本的功能都是正确的。

# Grading Policy

## 扩展功能实现 15 pts

- (5 pts) 以下功能择一实现：
  - 实现坚果墙，并且实现随血量变化的贴图与动画。
  - 实现铁桶僵尸，实现铁桶僵尸随血量变化的贴图与动画。
  - 或者任意自创特色植物/僵尸，要求其功能拥有随阶段变化的贴图与动画。
- (10 pts) 以下两种功能择一实现：
  - 实现樱桃炸弹，实现爆炸动画并且移除炸毁的僵尸。
  - 实现撑杆跳僵尸，实现撑杆跳功能以及不同阶段的僵尸行为表现及其贴图动画。

# Grading Policy

## 代码规范 (节选，详细代码规范见project1 文件）（0 pts ~ -20 pts）

- 编译没有来自于你自己的代码的 **warning** 。
- 请注意代码文件的include包括CMake文件的引用需要使用相对路径而非绝对路径。
- 至少添加一个新的cpp文件以及hpp文件。按类别把文件分在不同的子目录里，并为每个子目录都创建 CMakeLists.txt ，以 add_subdirectory 的方式添加进项目。
- 对于相似类的类似功能（各种种子的 OnClick 、各种僵尸的 Update等 ），不应复制代码，而是应该在基类里统一实现。
- 正确使用枚举 (enum) 类型，并且应该使用限定作用域的枚举类型（即 enum class ），除非有特殊原因（请解释）。不应使用神秘数字 **(magic numbers)** 或字符串来完成本该由枚举类型完成的功能。
- 通常情况下，所有数据成员几乎都属于“实现细节”，应当设置为 private 。注意：一些需要被子类访问的数据成员应该设置为 protected 。但是你不能滥用 protected ，因为 protected 成员的封装性实际上和 public 是一样弱的。
- 变量应在即将使用时才被声明，特别是不应随意使用全局变量（常量除外）。
- 你应当正确判断存储游戏内容对象所使用的数据结构（将实现写在**Containers**文件夹中），你的数据结构应该从**CS101A**的数据结构前**5**节课中挑选，并且自己独立实现该数据结构，而非使用**STL**。