1. Introducción

Este proyecto está planeado para la evaluación del tercer parcial. Dentro de este proyecto se emplearán la mayoría de conceptos aprendidos durante todo el curso. Este pequeño reporte tiene como propósito explicar un poco de cómo se resolvieron algunos problemas durante la creación del mismo.

2. Primer problema : La lengua

Cuando me encontré con este reto al inicio no sabía cómo resolverlo. Mi manera de solucionarlo fue crear una *variable estática* dentro de mi función **DibujarSerpiente** denomida *bod* la cual se encarga de variar entre 1 y 0. Lo que implica que cuando el timer ocurre una vez, se dibuja, y a la siguiente ocasión ya no lo hace.

También hacer mención que la lengua se dibujó con la función `Polygon()` empleando el tipo de dato `POINT`.

```
void DibujarSerpiente(HDC hdc, const PEDACITOS *serpiente)
{
static int bod = 0;
//Codigo para pintar y dibujar todo
switch(serpiente[i].dir){ //Cuadro de cabeza
    case N:
        if(bod==0)
            //Dibuja lengua
        else
            bod = 1;
        break;
    ...
}
}
```

3. Segundo Problema: Las Letras

Esto fue una de las cosas más sencillas de hacer, pero las que más tiempo llevo poder realizar. Ya que mi manera de crear las letras, en lugar de usar de nuevo `Polygon()` hice uso de un tipo de dato llamado **LETRA**. El cual se quedaba de forma .estático"por así decirlo, hablando del tamaño del dato que necesitaba para poder realizar las letras. Un poco del código necesario para poder crear las letras (solo son las posiciones).

```
LETRA *CrearNombre(RECT rect)
{
        int i=0;
```

```
4           LETRA *letras = NULL;
5           letras = (LETRA *) malloc(sizeof(LETRA) * TAMNOM); //
                TAMNOM es una constante
6           if(letras == NULL){
7                   MessageBox(NULL,"Sin memoria","Error",MB_OK |
                        MB_ICONERROR);
8                   exit(0);
9           }
10          while(i<4){ //palo de J primeros 4
11                  letras[i].pos.x = 5 + 7;
12                  letras[i].pos.y = i+3 + 3;
13                  i++;
14          }
15  }
```

En realidad algunas veces si se podía usar ciclos, sobre todo cuando eran bloques colocados de manera consecutiva en X ó en Y, De no ser así, tuve que colocarlos uno a uno.

4. Tercero: Una vez con posiciones X,Y. Faltaba dibujarlo...

Esto se hizo con una pequeña función llamada $DibujarNombre$ la cual recibía toda mi información con las posiciones de mis letras (cubitos).

```
1   void DibujarNombre(HDC hdc, const LETRA *letras){
2   {
3       //Codigo para el color
4      for(int i=0; i<TAMNOM; i++){
5           hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);
6           RoundRect(hdc, letras[i].pos.x * TAMSERP, letras[i].pos
                .y * TAMSERP,
7                   letras[i].pos.x * TAMSERP + TAMSERP, letras[i].
                        pos.y * TAMSERP + TAMSERP, 5, 5);
8           SelectObject(hdc, hOldBrush);
9       }
10  }
```

5. Cuarto: La Colisión con el Nombre

El siguiente pensamiento me llevó a la solución:
Tengo que ver por cada cubito de mis letras si no ha chocado en especificamente con la cabeza, o incluso también, con el segundo pedacito.

En ese momento también me di cuenta de un pequeño bug que me podría venir al

querer resolver la colisión, pues cuando yo comiera para crecer y esta se generara a un lado , técnicamente la cabeza de la serpiente ya no estaría en la misma posición que la letra.

```
int Colisionar(PEDACITOS *serpiente, int tams, const LETRA *
    letras, ENEMY *enemigo, PEDACITOS *snakeBackup){
{
    //Codigo restante
    while(i<TAMNOM){
        if((serpiente[tams-1].pos.x == letras[i].pos.x &&
            serpiente[tams-1].pos.y == letras[i].pos.y) || (
            serpiente[tams-2].pos.x == letras[i].pos.x &&
            serpiente[tams-2].pos.y == letras[i].pos.y)){
            stats.colision++; //Estadisticas, no explicado en
                este reporte
            state = 1;
            PlaySound("Sounds\\death.wav", NULL,
                SND_FILENAME | SND_ASYNC); //
            while(j<tams){ //No explicado en este reporte
                    serpiente[j].pos.x = snakeBackup[j].pos
                        .x;
                    serpiente[j].pos.y = snakeBackup[j].pos
                        .y;
                    serpiente[j].tipo = snakeBackup[j].tipo
                        ;
                    serpiente[j].dir = snakeBackup[j].dir;
                    j++;
            }
            return 1;
        }
        i++;
    }
}
```

6. Quinto: Bug de comida en letras

Si la comida se genera en una posición aleatoria..., solo basta con verificar si está en la posición de alguna de mis letras y si eso se cumple, la regreso a verificar eso.

Situandose en la función $WndProc$ justo en el `timer`...

```
com.pos.x = rand()% rect.right / TAMSERP;
com.pos.y = rand()% rect.bottom / TAMSERP;
while(i<tams){
    if(com.pos.x == serpiente[i].pos.x && com.pos.y ==
        serpiente[i].pos.y){
```

```
5           com.pos.x = rand()% (rect.right-15) / TAMSERP;
6           com.pos.y = rand()% (rect.bottom-15) / TAMSERP;
7           i=0; //Regresate a revisar
8       }
9   i++;
10  }
```

7. Sexto: Enemigos, Niveles y Velocidad

Los enemigos podrían aparecer dependiendo del nivel en el que estés. Si estás en el nivel 1, hay 1 enemigo, y así sucesivamente. Solo hace falta verificar el tamaño de la serpiente para poder determinar si subió o no de nivel. Y luego al mismo tiempo puedo solucionar la velocidad del juego
Situado en la función $AjustarSerpiente$, y en el caso cuando la serpiente $CRECE$.

```
1   switch(comida){
2       case CRECE:
3            if(*tams == 8 || *tams==11 || *tams==14 || *tams==18
                 || *tams==22 || *tams==25 || *tams==30 || *tams==35
                  || *tams==40){
4             PlaySound("Sounds\\levelup.wav", NULL, SND_FILENAME
                  | SND_ASYNC);
5             levelUPDW = 1;
6             nivelAct++;
7             nivelValue-=10;
8       }
9   }
```

Explicando:
La variable `tams` es el tamño de la serpiente. Entonces, cuando come y llega a cierto tamaño entonces se dice que subió de nivel. En el caso de que eso ocurra, se reproduce un sonido (cuando decrece, muere y come sucede lo mismo pero con diferentes sonidos)..., cuando eso ocurre entonces se dice el nivel aumenta ($nivelAct++$), ya que esa variable controla el nivel actual del juego. $nivelValue$ controla el valor del $timer$ respecto al nivel en el que nos encontramos.

Nota: La variable $levelUPDW$ sirve para saber si el usuario sube de nivel y hacer otro par de cosas como modificar el timer, crear más enemigos, etc.

Los **ENEMIGOS** son algo dinámico, es algo que va estar cambiando durante el juego. Su función es la siguiente:

```
1   ENEMY * CrearEnemigo(RECT rect)
```

```
{
        ENEMY *enemigo = NULL;
        enemigo = (ENEMY *) realloc(enemigo,sizeof(ENEMY) *
            nivelAct);
        if (enemigo==NULL){
                MessageBox(NULL,"Sin memoria","Error",MB_OK |
                    MB_ICONERROR);
                exit(0);
        }
        enemigo[0].pos.x = rand()% (rect.right / TAMSERP);
        enemigo[0].pos.y = rect.bottom / TAMSERP / 2;
        return enemigo;
}
```

De tal manera que por defecto hay un enemigo. Conforme se sube de nivel, tal cómo se mencionó más arriba, saldrán más enemigos al verificar la variable $levelUPDW$.

```
ENEMY * AjustarEnemigo(ENEMY *enemigo, RECT rect){
        enemigo = (ENEMY *) realloc(enemigo, sizeof(ENEMY) *
            nivelAct); //Crear mas enemigos
        if(enemigo == NULL){
                MessageBox(NULL,"Sin memoria","Error",MB_OK |
                    MB_ICONERROR);
                exit(0);
        }
        enemigo[nivelAct-1].pos.x = rand()% (rect.right /
            TAMSERP);
        enemigo[nivelAct-1].pos.y = 0;

        return enemigo;
}
```