

# ALGORITMOS Y ESTRUCTURAS DE DATOS

## Guía 3: Primero Pasos en C++



### Primero Pasos en C++

1. Se pide crear una clase Blob que nos permita modelar un blob con las siguientes características. Velocidad de desplazamiento, dirección, posición, radio de percepción de alimento (en unidades de medida), grupo etario, probabilidad de muerte.
2. Considere crear una clase Food. Debe conocer su posición, y el tamaño en ancho y alto.
3. Agregar a la clase una función que permita inicializarlo. ¿Dicha función tiene un nombre particular? ¿Cómo conviene inicializar al Blob?
4. ¿Se pueden agregar más funciones de inicialización? ¿Qué parámetros considera?
5. Agregar a la clase una función miembro que reciba una referencia a un Blob (¿por qué una referencia y no un Blob?) e indique si el Blob recibido se encuentra dentro del rango de colisión. ¿Cómo puede replicar esto con un objeto Food? ¿Qué otro parámetro debería conocer del Blob?
6. Agregar a la clase una función miembro para mover el Blob.
7. Agregar a la clase funciones miembro para modificar sus parámetros.
8. De las funciones agregadas, evaluar cuáles deberían ser públicas y cuáles privadas. Modificar el código apropiadamente.
9. ¿Las variables miembro definidas en (1) deberían ser públicas o privadas? ¿Por qué?
10. Crear una función que tome un Blob y lo grafique.

## Take Home:




### 11. Simulación de BLOBS:



Se busca simular el comportamiento de un grupo de Blobs que interaccionan entre sí y deben cumplir con una serie de reglas. Se inicia con una cantidad ingresada por el usuario en la GUI.

- Tick  $t = 0$ . Los blobs se encuentran en sus posiciones iniciales, con sus respectivas direcciones iniciales.
- Tick  $t = 1$ . Los blobs trazan un radio de **smellRadius** unidades y se fijan si hay alimento dentro de dicho radio. De encontrar alimento, modifican su orientación de forma que su nueva orientación sea hacia el que tienen más cerca. Se repite esta operación para todos los blobs y luego se mueven una unidad, en la dirección en que están apuntado.
- Tick  $t = 2$ . Los blobs repiten el algoritmo de b y continúan moviéndose.
- Se trata de un espacio toroidal, por lo que, si un blob sigue por fuera de los límites del mapa, aparece por el límite inmediatamente opuesto. Por ejemplo, si el blob se mueve una unidad a la izquierda y no hay más posibilidad de moverse hacia la izquierda continúa con su trayecto pero apareciendo por la derecha.
- Así se sucede la simulación hasta que el usuario decide cancelarla.
- Características de los blobs:
  - El Blob posee una posición y una dirección de movimiento (**BlobMovement**), cada determinado tiempo (medido en **ticks**, unidad de tiempo para actualizar) debe modificar su posición actual acorde a dicha dirección.
  - Posee también velocidad: deberá considerarse que el programa puede ejecutarse con dos modos de funcionamiento:
    - En el **modo 1** los blobs en todo momento comparten la misma velocidad. El usuario setea la velocidad con dos valores:
      - Velocidad máxima medida en  $px/tick$  (elegida antes de que inicie la simulación).

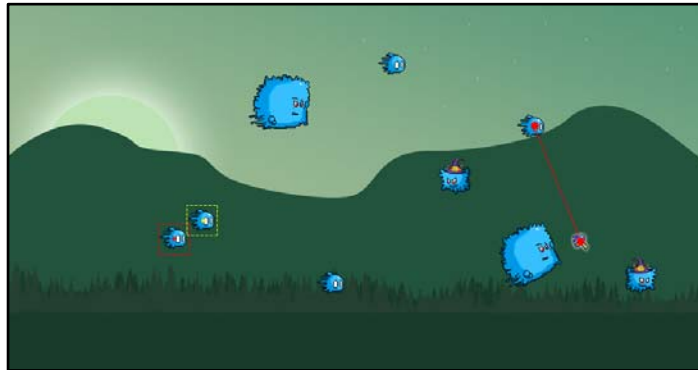
- **Velocidad porcentual** entre 0% y 100% (parámetro de simulación).
  - La velocidad total es la velocidad máxima multiplicada por la velocidad porcentual.
  - En el **modo 2**, cada blob tiene una velocidad máxima distinta a la de los demás. Estas se eligen aleatoriamente entre 0 y un valor máximo elegido por el usuario antes de que inicie la simulación. Todos los blobs comparten la velocidad porcentual.
- iii. Las direcciones iniciales de los blobs son calculadas de forma aleatoria. Luego, durante la simulación, la dirección cambia debido a varios fenómenos:
- **Fenómeno de BlobFeeding:** siguiendo el punto “b” explicado arriba, cuando el blob detecte que un alimento se encuentra a una distancia radial inferior a **smellRadius**, modifica su dirección de forma de dirigirse hacia él.
  - **Fenómeno de BlobMerge:** Los detalles de este fenómeno se incluyen en el ítem “iv” abajo.
- iv. El ciclo de vida de los Blobs tiene tres grupos etarios:

		
<b>BabyBlob</b>	<b>GrownBlob</b>	<b>GoodOldBlob</b>

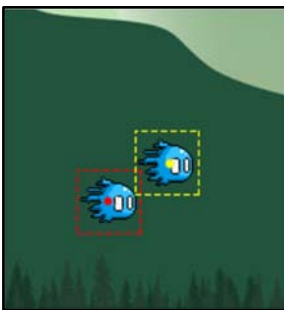
- Nacen en un primer estado denominado **BabyBlob**.
- Cuando colisionan dos o más **BabyBlob**, se unen para formar un **GrownBlob** (se considera que “colisionan” cuando se superponen sus bitmaps).
- De la misma manera, la colisión de dos o más **GrownBlob** hace que se unan para formar un **GoodOldBlob**.
- El proceso de fusión de blobs se denomina **BlobMerge**.
- Como resultado de un **BlobMerge**, la nueva dirección se calcula como la suma del promedio entre los Blobs participantes del proceso y número aleatorio entre 0 y **randomJiggleLimit** (un parámetro de simulación entre 0 y 360 que se detalla más adelante).
- La velocidad resultante es el promedio de la velocidad de los blobs fusionados.
- Cuando la colisión no es entre blobs del mismo grupo etario, o cuando ya no pueden evolucionar por ser todos **GoodOldBlob**, los Blobs pueden cruzarse sin interferirse, y darse un **BlobSaludo** mientras siguen su recorrido.

- g. En todo instante debe haber una cantidad de comida definida por un parámetro del usuario que denominamos **foodCount** ubicada de forma aleatoria. Al chocar con la comida, el Blob se alimenta, e internamente cuenta la cantidad de comida ingerida (proceso que denominaremos **BlobFeeding**).
- h. Según el estado de evolución, cuando llega a una cantidad determinada, se produce el milagro del **BlobBirth**, aparece un nuevo BabyBlob, y al existente se le reinicia el contador de alimento ingerido. En cada tick se repone el alimento, de forma que siempre hay **foodCount** alimento.
- i. La cantidad de alimento para la formación de un nuevo Blob es 5, 4, 3 para **BabyBlob**, **GrownBlob** y **GoodOldBlob** respectivamente.
- j. Cada tick puede producir una **BlobDeath**, el fin inesperado en la vida de un blob: todos tienen una **probabilidad de muerte** por tick. Al inicio de cada tick, para cada blob se sortea un número random entre 0 y 1. Si el número es menor a la probabilidad de muerte, el blob se muere. Todos los blobs dentro del mismo grupo etario tienen la misma probabilidad de muerte. Estas 3 probabilidades son parámetros de simulación.

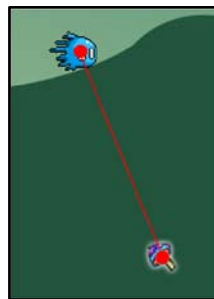
# Graphic User Interface



- a. La cátedra provee sprites para cada uno de los Blobs, la comida y el fondo de pantalla de la simulación.
- b. La simulación tiene parámetros que se deben poder modificar previo y durante la simulación, salvo excepciones indicadas. Son los siguientes:
  - a. Modo de simulación: 1 o 2 (no debe ser modificado durante la simulación).
  - b. Cantidad inicial de blobs (no debe ser modificado durante la simulación).
  - c. Velocidad máxima de los blobs. Cambia su funcionamiento según sea modo 1 o modo 2.
  - d. Velocidad porcentual de los blobs. (se sugiere utilizar un slider)
  - e. **smellRadius**: Radio de detección de la comida desde el centro del Blob.
  - f. **randomJiggleLimit**: Valor aleatorio a sumar para el cálculo de la dirección.
  - g. Probabilidad de muerte de cada grupo etario  $\in (0,1)$
  - h. **foodCount**: Cantidad de alimento.



Para las colisiones, los blobs no se consideran puntuales. Hay una colisión cuando los bitmaps de dos Blobs del mismo grupo etario se superponen. En la figura superior se muestran dos blobs con el borde de sus bitmaps en línea punteada. Como se superponen, están colisionando.



Los Blobs y la comida se consideran puntuales para todos los casos excepto las colisiones. La distancia entre dos elementos (Blob-Blob o Blob-Comida) se mide entre las posiciones centrales de ambos elementos, y *no desde el borde de sus bitmaps!*