Game Loop

Professors de VJ

Framerate

Framerate

- Nombre de frames (renders) per segon (fps).

Els jocs busquen:

- Framerate consistent.

Framerates típics:

- 30 fps
- 60 fps
- 120 fps

Temps per frame = 1 / framerate

Framerate

- Jocs de reacció ràpida (twitch): >= 30 fps
  - Millor punteria, es veuen els moviments abans.
- Màxim framerate útil → El del monitor.
  - Difícil d'aconseguir ← Problemes de sincronització
- Sentit de presència: mínim de 10 fps.
  - Però és força questionable.

Lag

Temps entre:

- Jugador efectua una acció.
- Efecte en el món del joc.

Lag massa gran:

- Trenca la causalitat.
- Dificulta la interacció, la punteria, ...
- Pot produir mareig.

És més fàcil ajustar-se a lag constant que a lag variable.

Game Loop

Component central de tot joc.

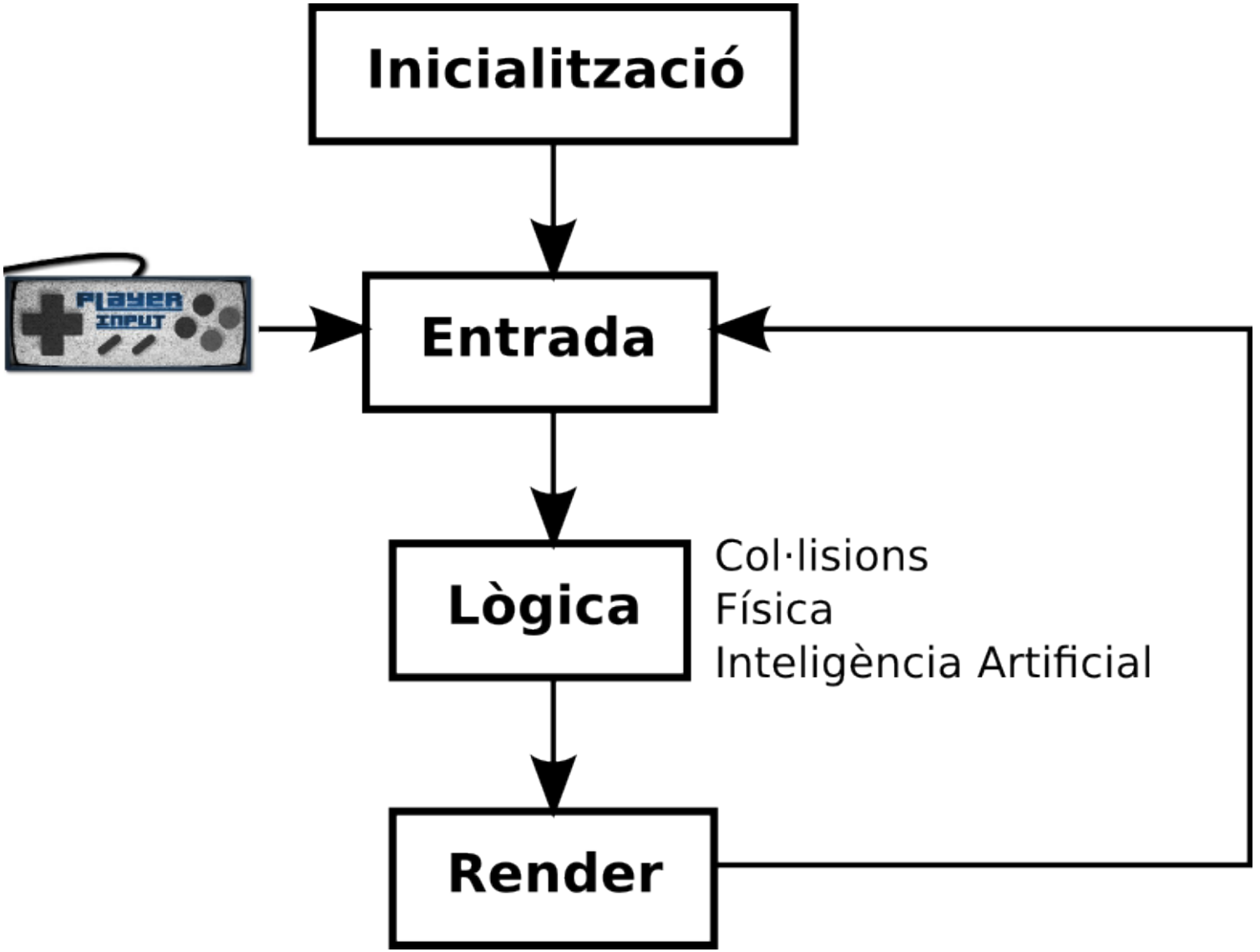
Característiques:

- Permet al joc funcionar encara que no hi hagi events externs.
- Seqüència d'accions a realitzar a cada frame.

Motivació

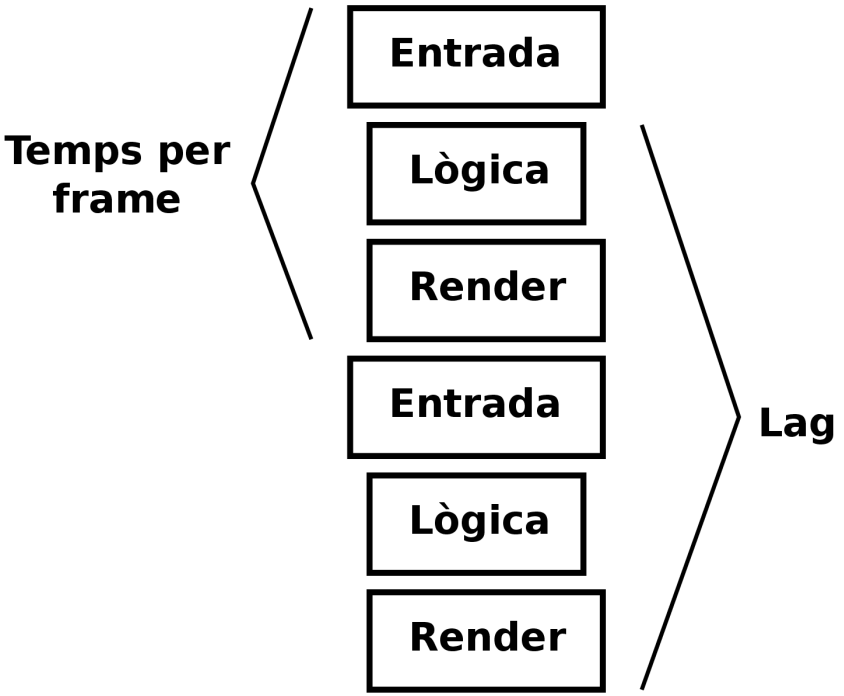
- Els jocs són un subconjunt de real-time programming.
- Relació determinista entre temps de les diverses tasques.

Game Loop bàsic



Lag - Framerate

El lag no és el temps per frame.



Lag màxim = 2 / framerate

Lag mitjà = 1.5 / framerate

Entrada (Player Input)

Dues opcions:

- Polling: Preguntar continuament.
- Event-driven: Només quan hi ha un event.

Polling

```
while ( true )
{
  if(e = obtindreEvent())
    tractarEvent();
  gameLoop();
}
```

- Cal crida “obtindreEvent” no bloquejant.
- O crida per esperar nou event.

Entrada (Player Input)

Event driven

- L'aplicació registra relacions entre events i funcions
  - Es produeix l'event → Es crida a la funció
- GLUT → Callbacks
  - glutIdleFunc
  - glutResizeFunc
  - glutKeyboardFunc
  - glutMouseFunc
- Internament la llibreria que fem servir:
  - Rep les interrupcions del SO.
  - Crida a la funció que es correspongui → Taula de traducció

Entrada (Player Input)

Com escollir:

- Polling és ineficient.
- Event-driven pur no permet establir relació de temps entre tasques → Contra la idea del Game Loop.
- Solució:
  - Event-driven per registrar les entrades.
  - Procés a l'etapa “Entrada” del Game Loop.

Objectes dinàmics

Els jocs tenen objectes dinàmics.

Causes:

- Entrada del jugador.
- Inteligència artificial.
- Física.

Conseqüències:

- Interacció, col·lisions, ...
- Depenent del temps de cada volta del bucle.

Solució:

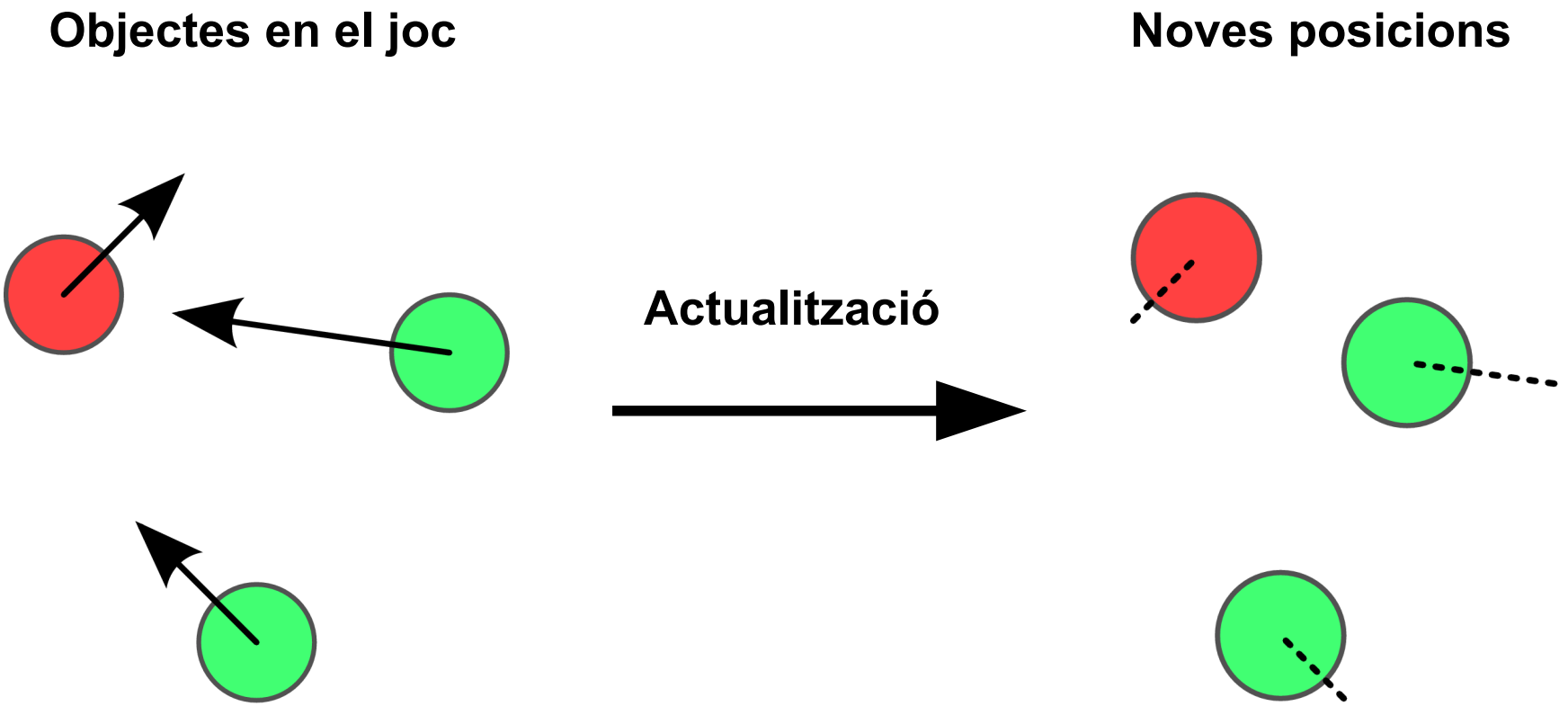
- Ús de velocitats.

Velocitats

Tots els objectes dinàmics tenen una velocitat assignada.

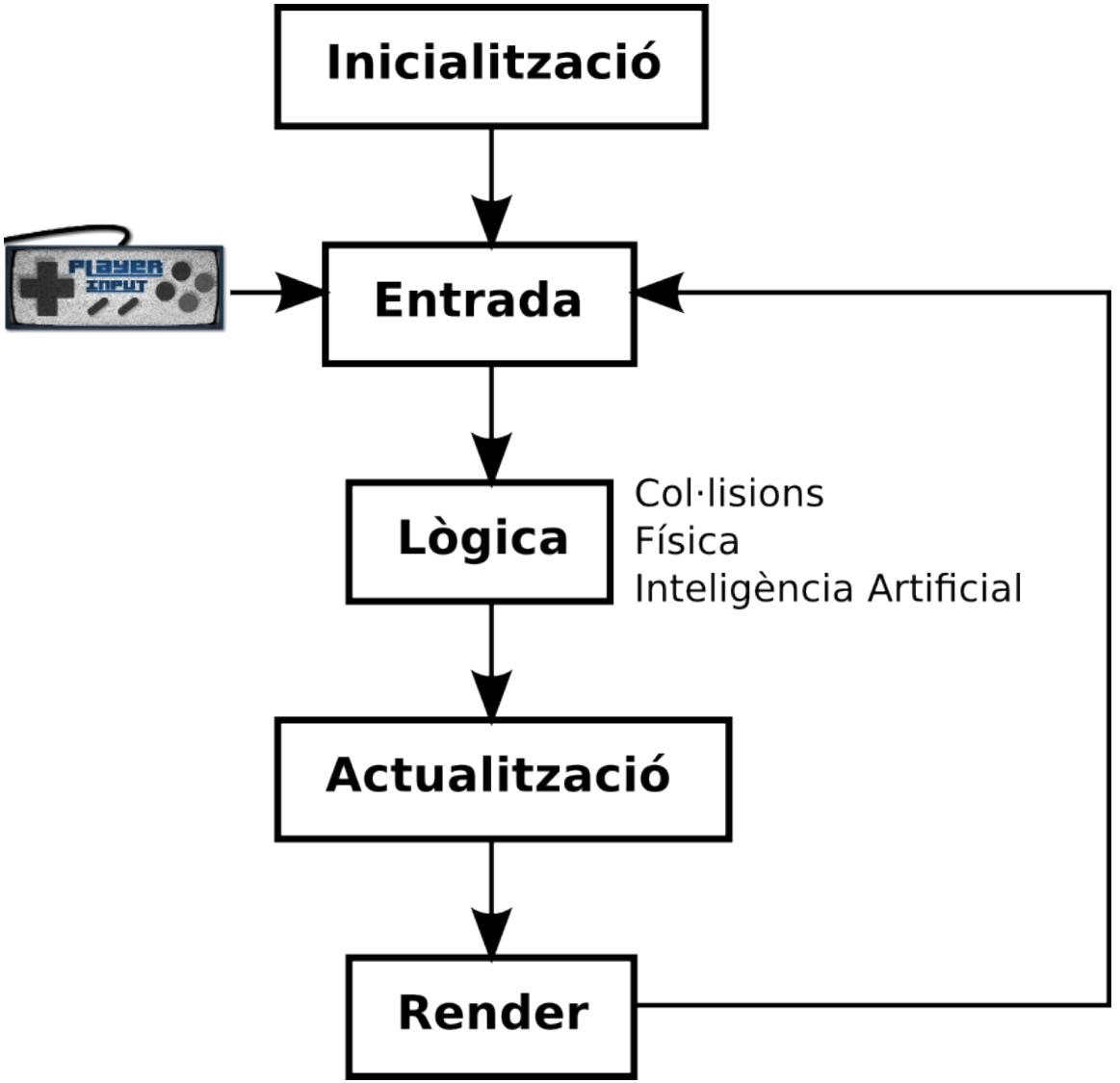
- En forma de vector:
  - 2D: Velocitat en x i y.
  - 3D: Velocitat en x, y i z.
- Actualitzada en els pasos de “Entrada” i “Lògica”.
- Les posicions dels objectes s'actualitzen en un nou pas.

Velocitats



Velocitats expresades en unitats per segon.

Game Loop


Inicialització
Player Input
Entrada
Lògica
Col·lisions
Física
Intel·ligència Artificial
Actualització
Render

Timing

Encara queda un problema per considerar:

- Velocitat variable del bucle:
  - Causes:
    - Hardware divers.
    - Temps assignat pel planificador multitasca.
  - Una altra rao per necessitar velocitats.
- Solució: Timing del bucle.

Timing

Cal fer servir “timers”.

- Ens permeten obtenir el temps transcorregut des d'un determinat moment.
  - Habitualment l'inici de l'execució.
- Millor resolució a la que podem aspirar:
  - Mil·lisegons.
- Depèn del SO:
  - GLUT: glutGet(GLUT_ELAPSED_TIME)
  - Temps en mil·lisegons des de l'inici del joc.
  - Temps transcorregut via diferències.

Timing

Tenim un temps esperat per frame (T_E):

- Si el framerate és F $\rightarrow$ T_E = 1000 / F mil·lisegons.
  - Si F = 50 fps $\rightarrow$ T_E = 20 ms
  - Si F = 60 fps $\rightarrow$ T_E = 17 ms

Com funciona:

- Calcular temps transcorregut T_T.

Timing

- Diferència entre temps: T_E-T_T:
  - Si és >= 0: Anem bé → Donem aquest temps extra al SO.
    - Bucle intern esperant el temps extra.
    - Fer un “sleep” (és la millor opció).
      - Android → Thread.sleep(milliseconds)
      - Windows → usleep(microseconds)
  - Si és < 0: No arribem → Necessitem accelerar.
    - Reduir qualitat dels gràfics (si és possible).
    - Interpolació → Aquí entren en joc les velocitats.
    - Millorar l'eficiència del joc.

Game Loop

```
loopTime = 0;
while(true)
{
  startTime = getCurrentTime();
  update(loopTime);
  render();
  endTime = getCurrentTime();
  loopTime = endTime - startTime;
  sleepTime = timePerFrame - loopTime;
  if(sleepTime > 0)
  {
    sleep(sleepTime);
    loopTime = timePerFrame;
  }
}
```

Game Loop - GLUT

Callbacks → glutIdleFunc(callbackFunc)

- Es crida sempre que es pot.

```
void idleFunc()
{
  endTime = glutGet(GLUT_ELAPSED_TIME);
  loopTime = endTime - startTime;
  startTime = glutGet(GLUT_ELAPSED_TIME);
  update(loopTime);
  render();
  // starTime, loopTime hauran de ser estàtiques
  // Cal inicialitzar startTime
}
```

Framerate variable no és un problema → No sleep.

Avançat

Cada tasca requereix ritmes d'actualització diferents:

- Gràfics: 30 fps
- IA: 10 fps
- ...

Si són múltiples podem reaprofitar el codi.

En cas contrari, hem de desacoblar.

Un thread per tasca.