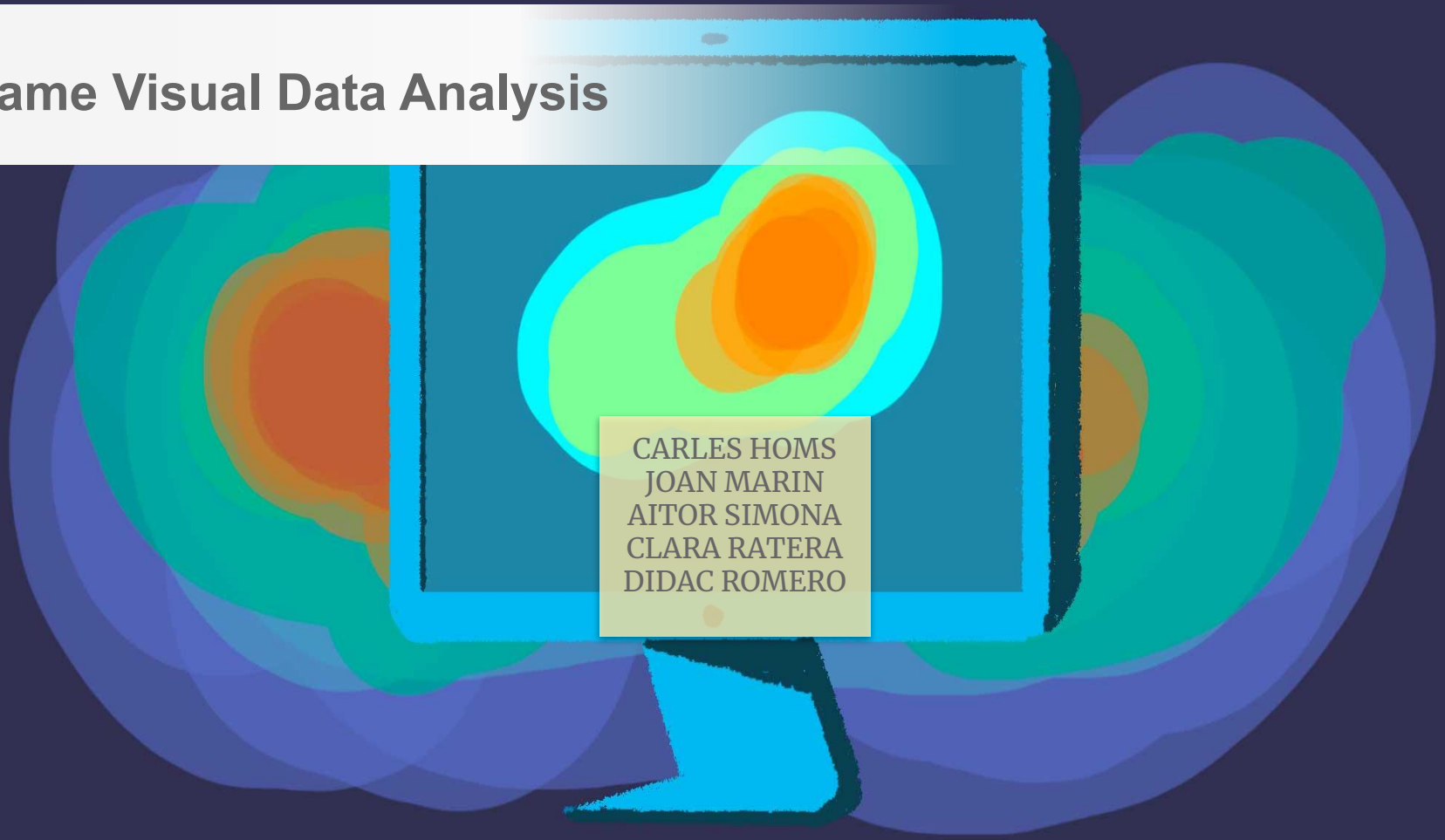
Game Visual Data Analysis



CARLES HOMS
JOAN MARIN
AITOR SIMONA
CLARA RATERA
DIDAC ROMERO

Player Simulation

- Simulation runs on startup, preparing both the player and the session.
- Script selects a Player of a fictional playebase each time the game runs.
- Creates a RegisterEvent IF first-time Player.
- Creates a SessionEvent for the chosen Player.

```
public struct session_data
{
    public int player_id;
    public int session_id;
    public System.DateTime start;
    public System.DateTime end;
}
public session_data currentSession = new session_data();
```

```
public struct player_data
{
    public int player_id;
    public string gender;
    public string first_name;
    public string second_name;
    public int age;
    public string country;
    public string test_group;
}
public player_data currentPlayer = new player_data();
```

```
public player_data[] playerbase = new player_data[10] {
    new player_data(1, "Agustin", "Sarin", 21, "M", "United Kingdom", "A"),
    new player_data(2, "Sol", "Coolbaugh", 14, "M", "India", "A"),
    new player_data(3, "Lasse", "Loepfe", 24, "M", "Germany", "B"),
    new player_data(4, "Ena", "Chivers", 35, "F", "France", "A"),
    new player_data(5, "Clarine", "Faubers", 42, "F", "United States", "B"),
    new player_data(6, "Ricard", "Pillosu", 71, "M", "Brazil", "A"),
    new player_data(7, "Lakenya", "Bushmaker", 57, "F", "China", "B"),
    new player_data(8, "Tyree", "Gustiuts", 12, "M", "Japan", "B"),
    new player_data(9, "Esteban", "Dacenzo", 17, "M", "Norway", "B"),
    new player_data(10, "Qiana", "Callejo", 66, "F", "Rusia", "A")
};
```

Event Data Structures

- **GenericEvent**: Base event class with only event_id
  - **RegisterEvent**: First-time player login event (player, miscellaneous player data)
  - **SessionEvent**: Player session event (player, session, start, end).
  - **GameEvent**: In-game event, saves generic data (timestamp, session, entity, pos, rot)
    - **HitEvent**: GameEvent with an added LifeRemaining data.

- **EventContainer**: Class which contains both a list of events and an object class to serialize them.
  - **EventList**: C# list for easy management and addition of events.
  - **EventWrapper**: Class to serialize, containing only an array assigned with all events inside the *EventList*.

```
// EVENT IDs
15 references
public enum event_types
{
    EVENT_REGISTER = 1,
    EVENT_SESSION,
    EVENT_WALK_POS,
    EVENT_AIR_POS,
    EVENT_KEY_POS,
    EVENT_JUMP,
    EVENT_ATTACK,
    EVENT_HIT,
    EVENT_DEATH,
    EVENT_SPAWN,
    EVENT_INV_START,
    EVENT_INV_END
}
```

```
// EVENT CLASSES
[System.Serializable]
4 references
public class GenericEvent
{
```

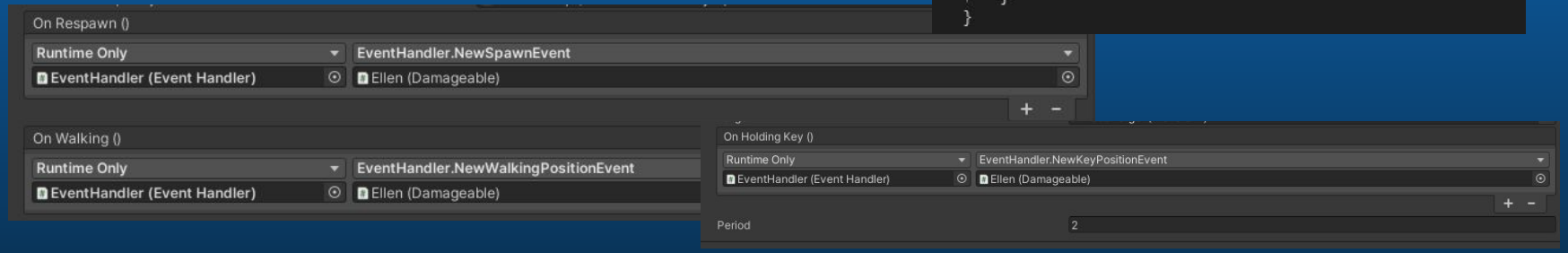
```
// GAME EVENTS
[System.Serializable]
33 references
public class GameEvent : GenericEvent
{
```

```
// EVENT CONTAINER
13 references
public class EventContainer<ContainerEventType>
{
    [System.Serializable]
    2 references
    public class EventWrapper<WrapperEventType>
    {
```

Calling Events

- Events are called on the relevant scripts for each ***GameObject***:
  - Player controller
  - Damageable
  - Transform follow
- Recurrent events have a timer editable from the inspector
- Event calls are shown and info set on the inspector

```
if(Time.time > nextKeyEvent)
{
    nextKeyEvent = Time.time + period;
    OnHoldingKey.Invoke();
}
}
```



Event Management

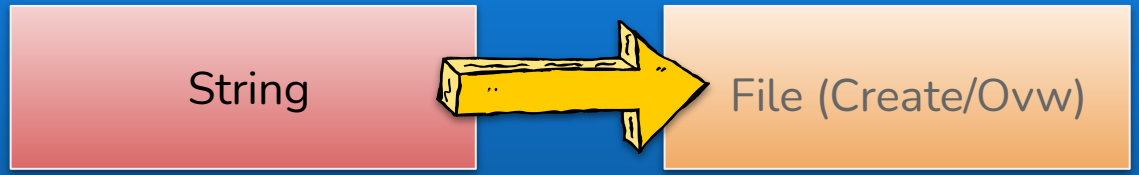
- Contains an *EventContainer* for each event type.
- ***On Startup**:* Loads all existing .csv files with JSON format into *EventContainers* lists using *FromJson* utility.
- ***Running**:* Receive *NewEvent* calls to load into respective *EventContainers* lists (Old .csv events + new ones).
- ***On End**:*
  1. Transform *EventContainers* event list into an array inside the container *EventWrapper* class.
  2. Serialize the *EventWrapper* object class into a string and Print it into .csv file, overwrite if existent.
     - Because we loaded the .csv events on startup, we're writing the old and the new events into the file.

```
// Players
if (registerEventContainer.Count > 0)
    DataSerializer.Overwrite(registerEventContainer.SerializeList(), directory + "Players.csv");
```

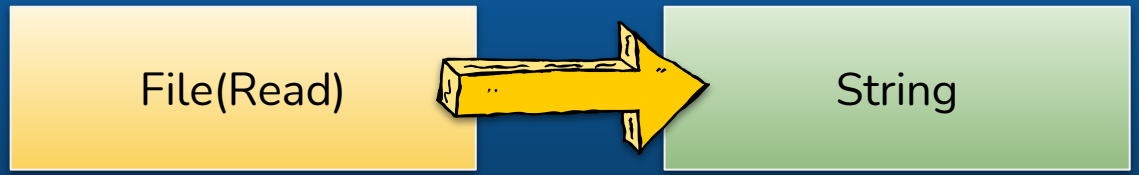
```
// Game Trigger Events
0 references
public void NewJumpEvent(Damageable character) // Position where jumped
{
    GameEvent gameEvent = new GameEvent(event_types.EVENT_JUMP, -1, character.gameObject.GetInstanceID(), character.transform);
    jumpEventContainer.Add(gameEvent);

    //string event_data = gameEvent.Serialize(true);
    //DataSerializer.Overwrite(event_data, directory + "Jumps.csv");
}
```

Serializer/Deserializer

- *Print / Overwrite:*



- *Read:*



Heatmap

2 approaches

- Cubes with gradient (Chosen) With adaptable density!
- Mesh that wraps the map with a shader

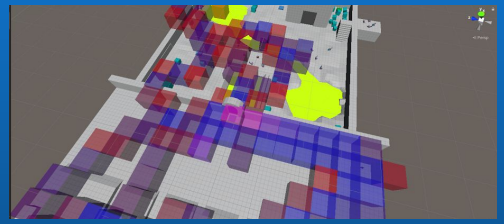
Final implementation:

- 2D Grid
- Vector of Events positions

We iterate the positions of the events and decide where in the grid they ++.

- Choose gradient value -> value/highest cube value

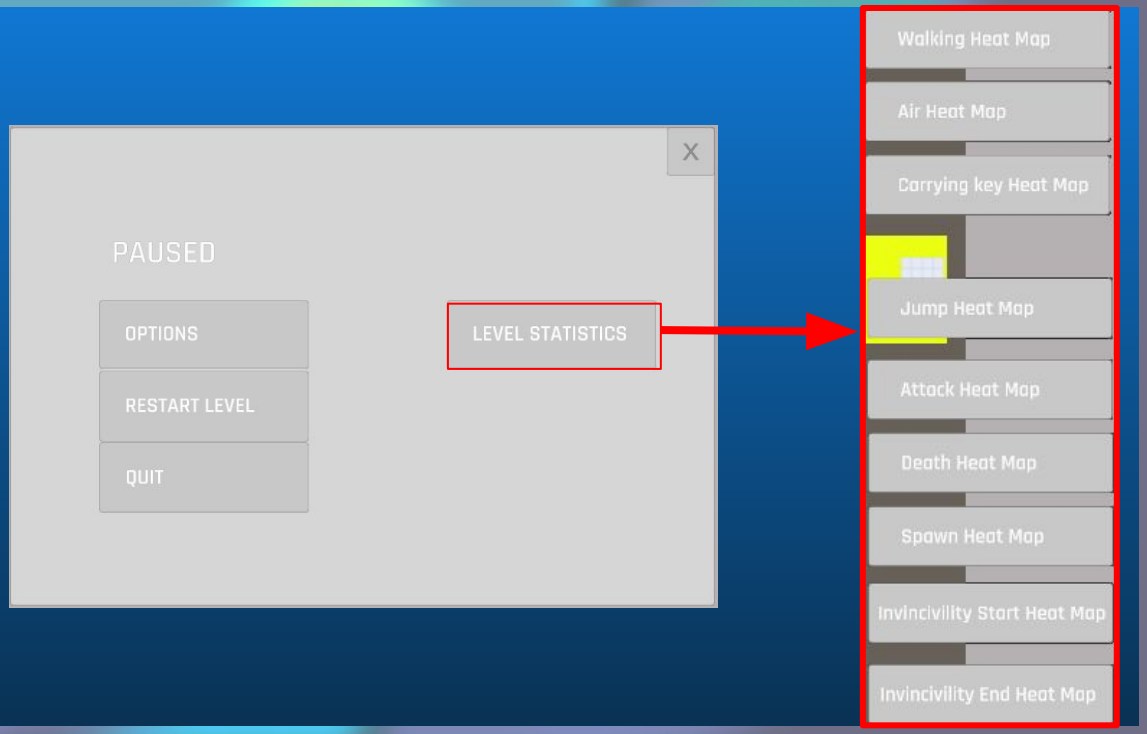
Cube size: 5



Cube size: 0.5

Heat Drawing

- Heatmaps available:
  - Walking
  - On air
  - Carrying a key
  - Jumping
  - Attacking
  - Death
  - Spawn
  - Invincible Start
  - Invincible End



Heat Drawing

Result: