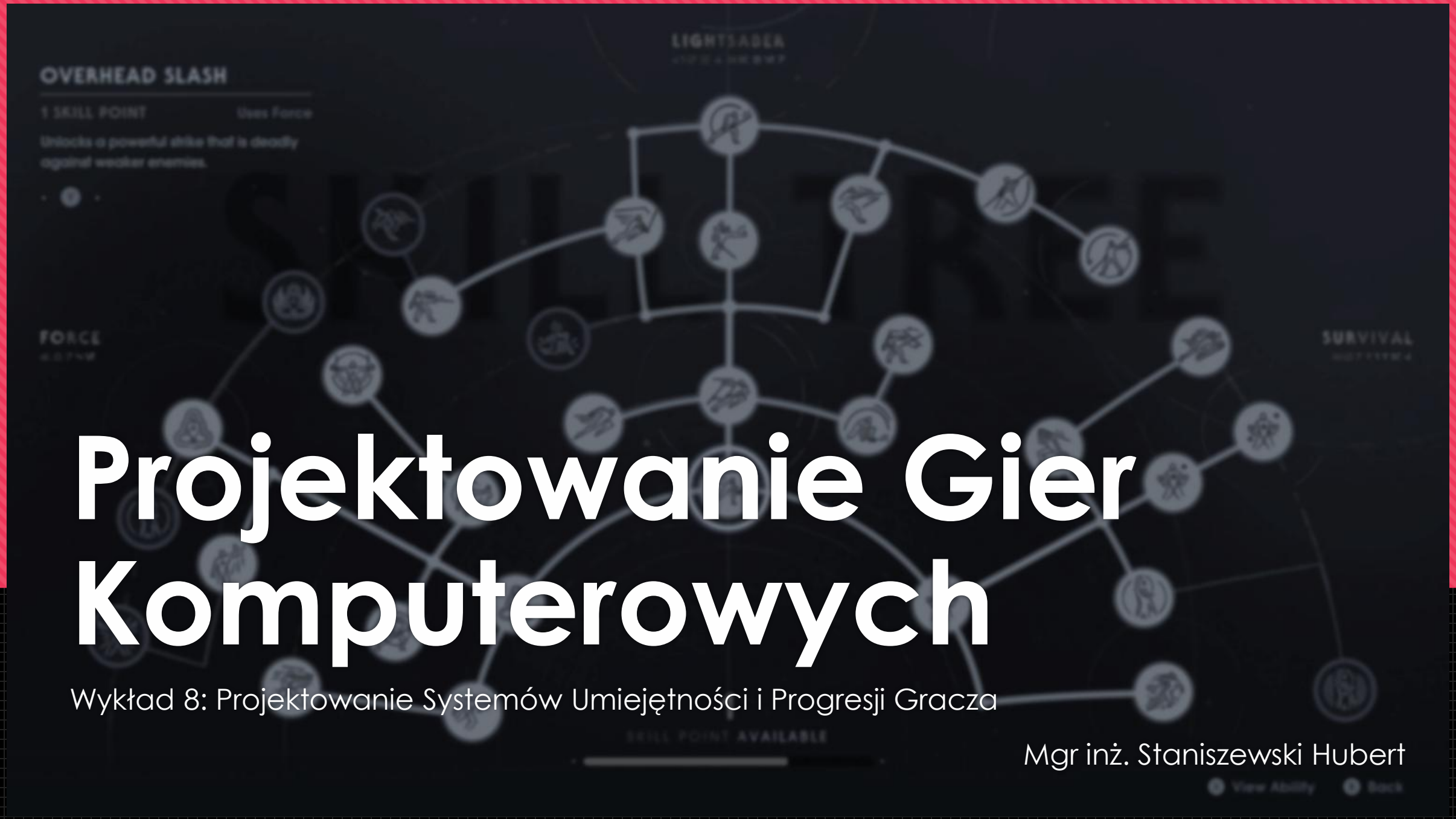


Projektowanie Gier Komputerowych

Wykład 8: Projektowanie Systemów Umiejętności i Progresji Gracza

Mgr inż. Staniszewski Hubert

Dlaczego progresja gracza ma znaczenie?

Systemy progresji napędzają zaangażowanie gracza. Nadają grze poczucie celu, nagrody za wysiłek i kontrolowanej transformacji postaci. Progresja może mieć charakter statystyczny, narracyjny lub mechaniczny – i zawsze powinna być powiązana z gameplay loopem.

Typy progresji postaci

- **Statyczna (predefiniowana):** gracz odblokowuje z góry zaplanowane umiejętności.
- **Modularna:** gracz wybiera spośród różnych opcji (np. klasy postaci).
- **Dynamiczna/otwarta:** drzewka z wieloma ścieżkami, specjalizacjami.

Rola systemów umiejętności w rozgrywce

System umiejętności:

- **Wzmacnia pętlę rozgrywki** (walka, eksploracja, crafting).
- Nadaje graczowi **poczucie wpływu** i personalizacji.
- Wprowadza **długoterminowe cele** i **strategiczne decyzje**.
- Pozwala różnicować styl gry.

Typowe formy drzewka umiejętności

- **Drzewko liniowe** – np. *God of War*: jedno wejście, wiele poziomów.
- **Rozgałęzione drzewko** – *Diablo II*: wybór ścieżki, ograniczona zmiana.
- **Siatka (grid)** – *Final Fantasy X*: odblokowywanie pól.
- **Sieć (web)** – *Path of Exile*: pełna swoboda, wysoka złożoność.
  Wybór formy wpływa na odbiór systemu i decyzje gracza.

Mechaniczne komponenty umiejętności

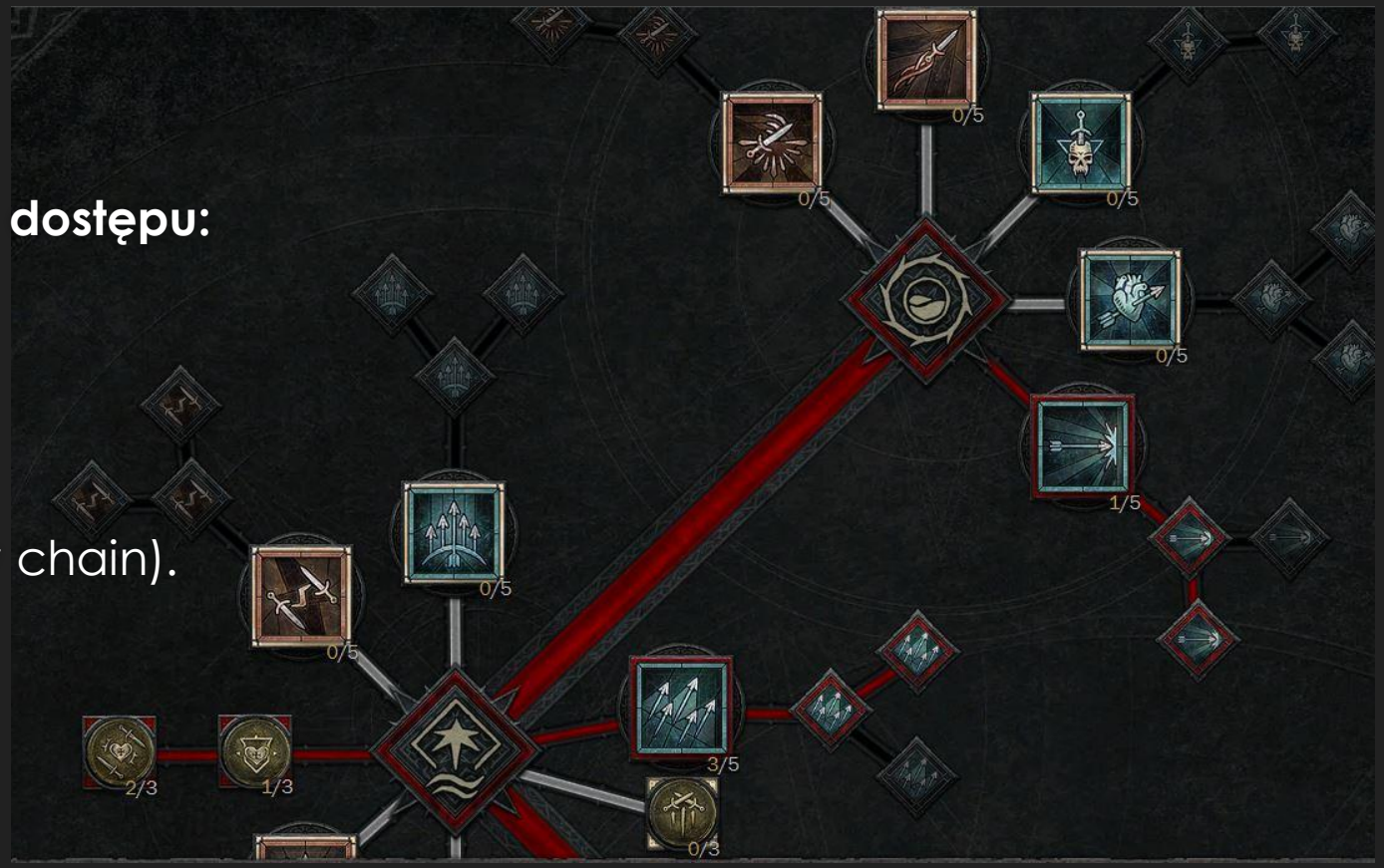
- Statystyki (zdrowie, mana, obrażenia).
- Zachowanie (szybszy reload, uniki).
- Aktywne działania (czary, ataki specjalne).
- Pasywne efekty (np. regen życia, odporności).

Równoważenie tych komponentów to klucz do satysfakcji gracza.

Warunki odblokowania – gating

Systemy progresji wymagają kontrolowania dostępu:

- **Poziom gracza** (XP).
- **Warunki fabularne** (questy).
- **Zasoby** (punkty umiejętności).
- **Umiejętności wcześniejsze** (dependency chain).



Punkty umiejętności i koszt rozwoju

Projektowanie kosztów:

- Niska cena = szybka rozbudowa, większa elastyczność.
- Wysoka cena = bardziej przemyślane wybory.
- Koszt progresywny (rosnący).
- Punkty resetu / cofnięcia (respec).

Systemy specjalizacji i klasy

Wielu graczy lubi się specjalizować – niektóre systemy powinny to umożliwiać:

- Role bojowe (tank, DPS, support).
- Style rozgrywki (agresywny vs taktyczny).
- Klasyczne archetypy RPG (czarodziej, wojownik, łotrzyk).

Resetowanie i przebudowa buildów

Czy gracz może zresetować drzewko?

- Brak opcji = większa waga decyzji, ale mniejsza elastyczność.
- Swobodny reset = większa eksploracja systemu.
- Kosztowny reset = zachęta do przemyślanych buildów, ale z opcją korekty.

Balans systemu umiejętności

Balans to jedno z największych wyzwań, typowe problemy to np.:

- „Jeden build rządzi” (dominacja).
- Bezużyteczne ścieżki (trap builds).
- Power creep.

Umiejętności

Reprezentacja umiejętności – co to właściwie jest?

Umiejętność w grze to **moduł funkcjonalny** o określonym zachowaniu i parametrach:

- Nazwa, koszt, cooldown.
- Typ (aktywny, pasywny).
- Efekt (obrażenia, leczenie, buff).
- Warunki aktywacji (input, stan gracza, cooldown).

Struktura klasy umiejętności (C++)

Przykładowwa struktura klasy umiejętności, która będzie podstawą dla każdej następnej umiejętności.

```
UCLASS(Blueprintable)
class USkillBase : public UObject
{
    GENERATED_BODY()

public:
    UFUNCTION(BlueprintNativeEvent)
    void Activate(ACharacter* Instigator);

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    float Cooldown;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    float ManaCost;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    TSubclassOf<UEffect> EffectClass;
};
```

System komponentowy – SkillComponent

Dobrym wzorcem jest trzymanie umiejętności w **komponencie postaci**:

```
UCLASS(ClassGroup=(Custom), meta=(BlueprintSpawnableComponent))
class USkillComponent : public UActorComponent
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere)
    TArray<TSubclassOf<USkillBase>> Skills;

    void ActivateSkill(int32 Index);
};
```

Wersja Blueprintowa – prosty system drzewka

W Blueprintach możemy:

- Stworzyć **Data Asset** typu **SkillData** z polami: nazwa, ikona, koszt, efekt.
- Zbudować logikę drzewa jako Widget, w którym umiejętnościami są przyciskiem.

Po kliknięciu:

- Sprawdzanie warunków (punkty, wymagane umiejętności).
- Aktywujesz umiejętność (dodanie jej do listy aktywnych).
- Odtwarzanie animacji / efektu.

Uwaga: Przechowywanie odblokowanych umiejętności => w **SaveGame** lub **PlayerState.**

Integracja z gameplayem

Umiejętności muszą:

- **Mieć efekt w grze** (np. spawn pocisku, zwiększenie statystyk).
- **Komunikować się z postacią** (np. zmiana animacji, ograniczenie ruchu).
- **Reagować na cooldown** – najlepiej przez system **TimerHandle**.

Warto Integrować umiejętności z:

- **Animation Montage** (np. atak z opóźnieniem).
- **Gameplay Tags** – do określania stanu („State.Stunned”, „State.Poison”)
- **EQS / AI** – by boty umiały używać umiejętności.

Tagi

Tworzenie tagów odbywać się może W kilka sposobów.

Sposób pierwszy:

- Dodanie w ustawieniach projektu.

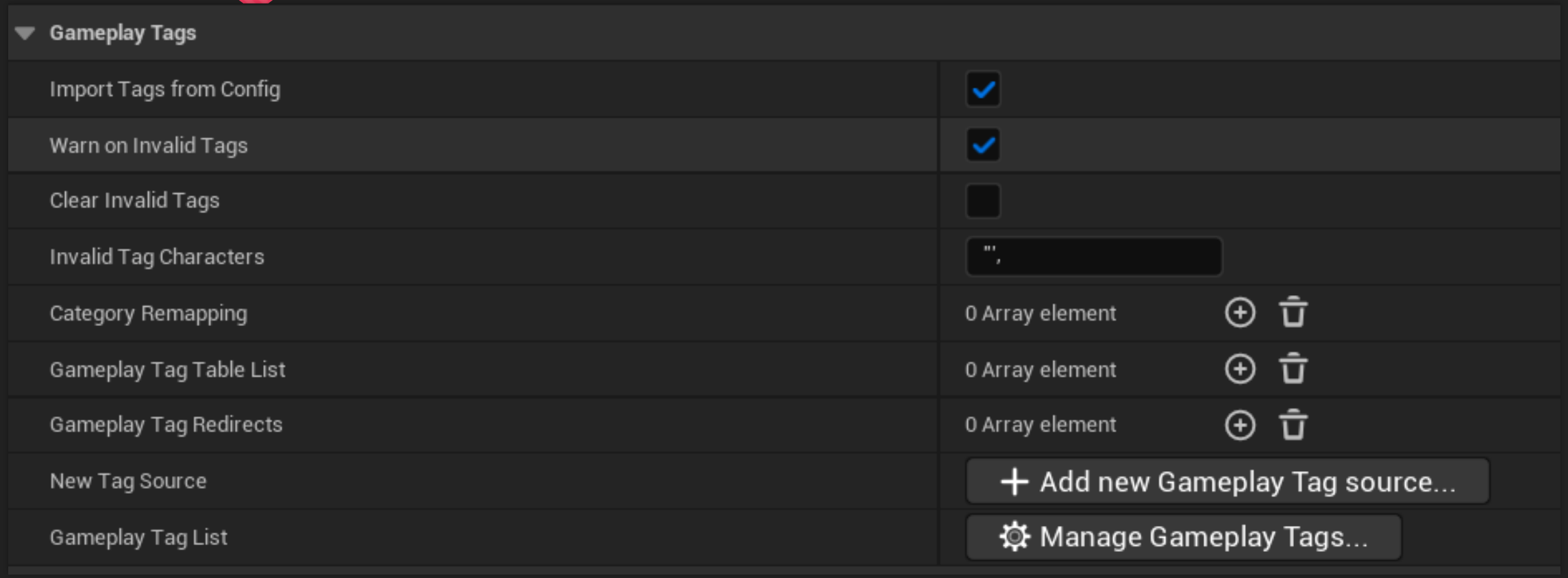
Sposób drugi:

- Dodawanie w Pliku .ini.

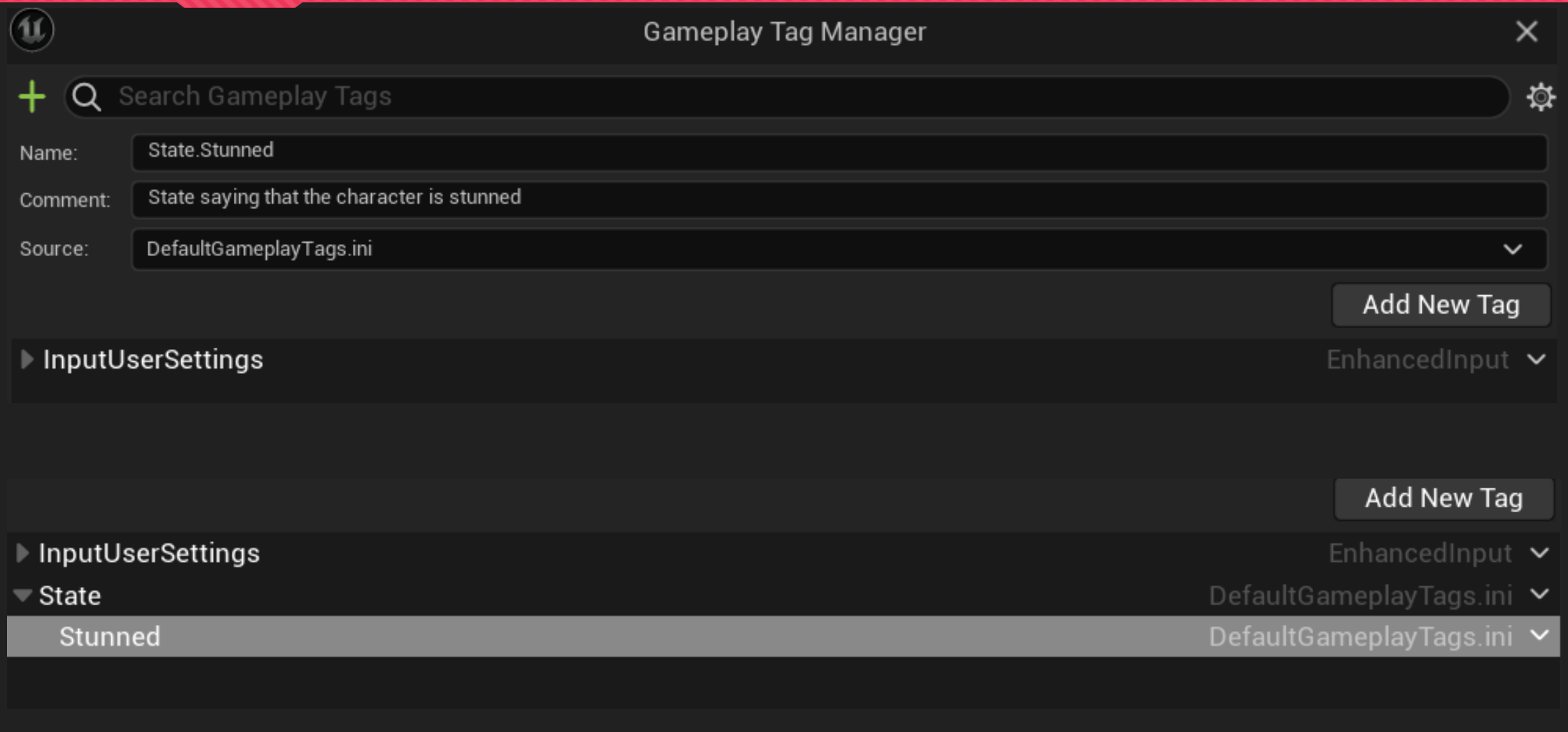
Sposób trzeci:

- Dodawanie przez DataTable.

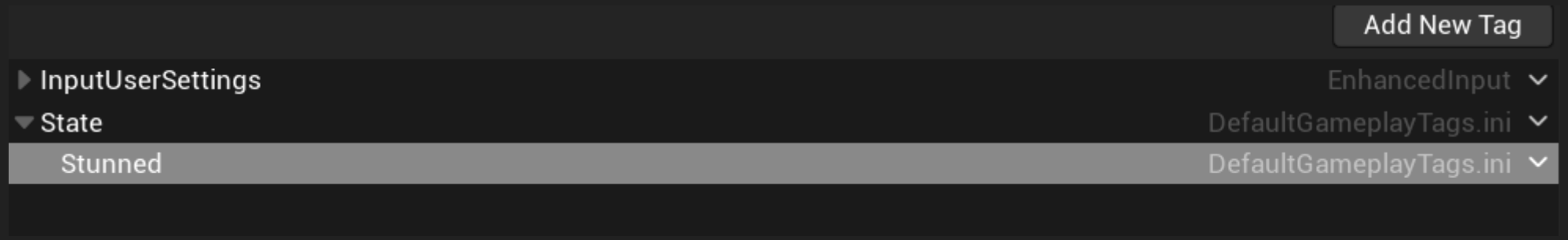
Ustawienia Projektu



Ustawienia Projektu



Ustawienia Projektu

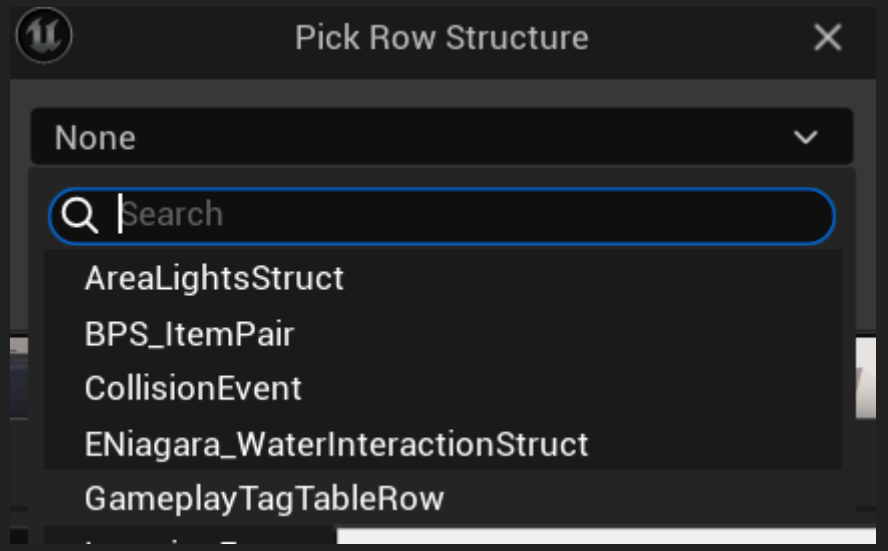


Pliki DefaultGameplayTags.ini

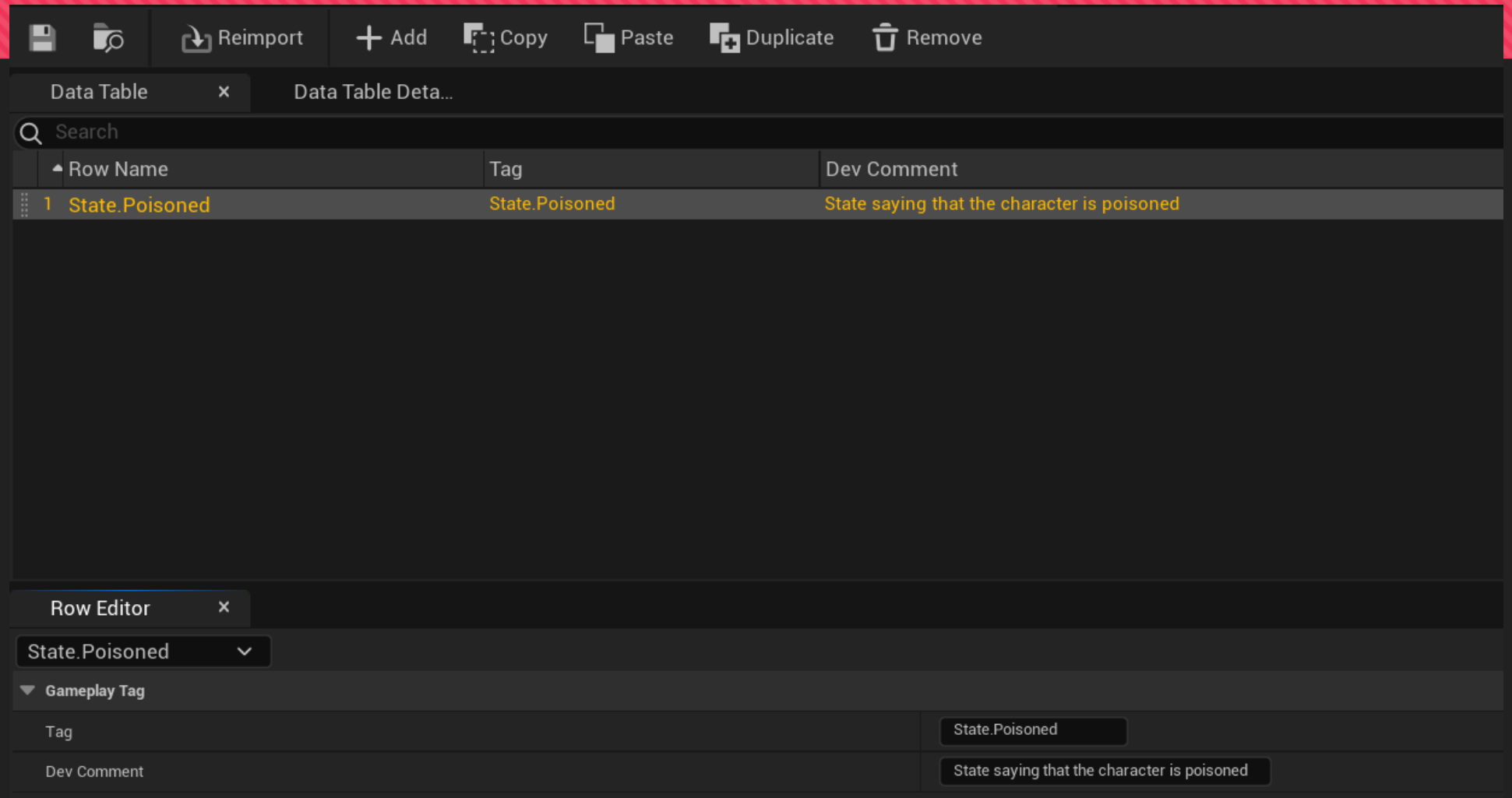
```
[/Script/GameplayTags.GameplayTagsSettings]
ImportTagsFromConfig=True
WarnOnInvalidTags=True
ClearInvalidTags=False
AllowEditorTagUnloading=True
AllowGameTagUnloading=False
FastReplication=False
InvalidTagCharacters="\"\',"
NumBitsForContainerSize=6
NetIndexFirstBitSegment=16
+GameplayTagList=(Tag="State.Stunned",DevComment="State saying that the character is stunned")
```

DataTable

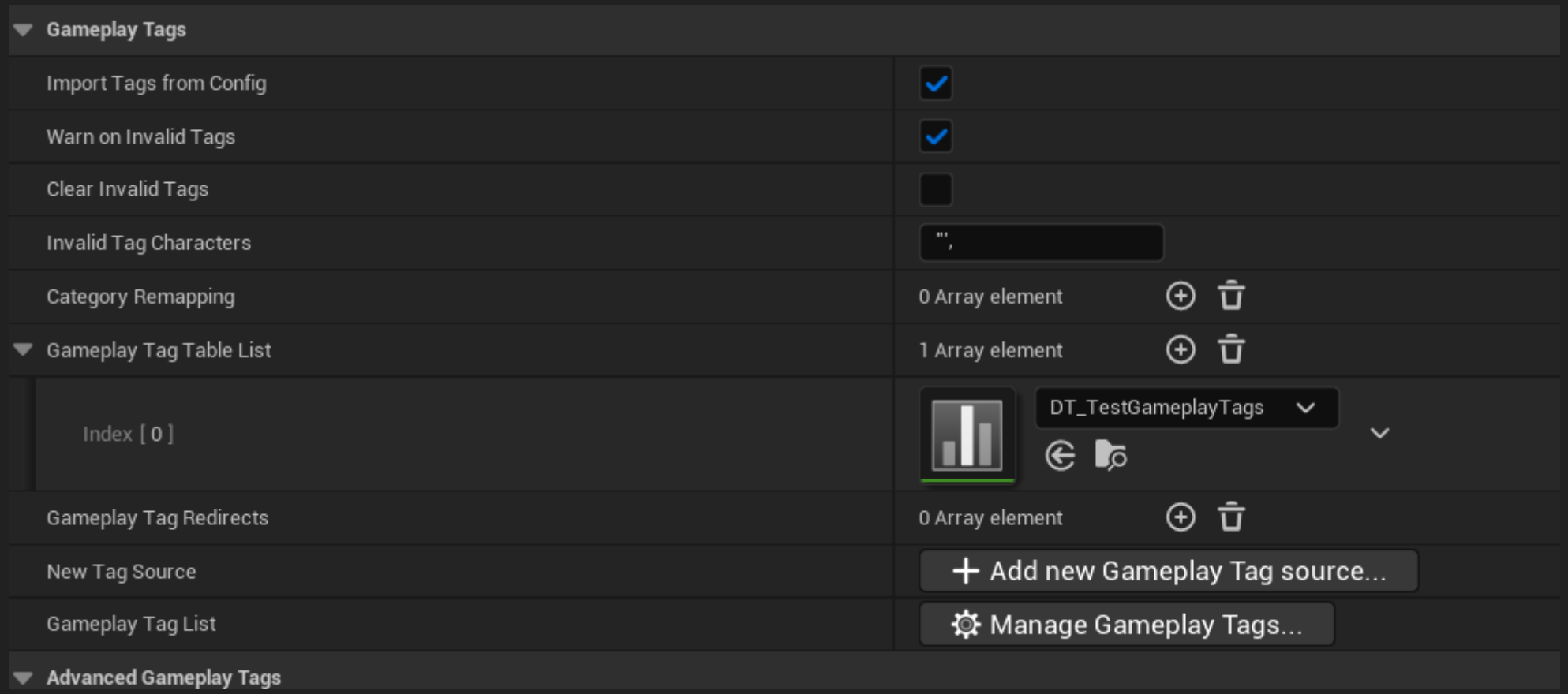

Media
Miscellaneous
Paper2D
Data Table
Imported spreadshee


Pick Row Structure
None
Search
AreaLightsStruct
BPS_ItemPair
CollisionEvent
ENiagara_WaterInteractionStruct
GameplayTagTableRow

DataTable


Reimport
Add
Copy
Paste
Duplicate
Remove
Data Table
Data Table Deta...
Search
Row Name
Tag
Dev Comment
1
State.Poisoned
State.Poisoned
State saying that the character is poisoned
Row Editor
State.Poisoned
Gameplay Tag
Tag
State.Poisoned
Dev Comment
State saying that the character is poisoned

DataTable


Gameplay Tags
Import Tags from Config
Warn on Invalid Tags
Clear Invalid Tags
Invalid Tag Characters
",
Category Remapping
0 Array element
Gameplay Tag Table List
1 Array element
Index [0]
DT_TestGameplayTags
Gameplay Tag Redirects
0 Array element
New Tag Source
Add new Gameplay Tag source...
Gameplay Tag List
Manage Gameplay Tags...
Advanced Gameplay Tags

Czy warto używać Gameplay Ability System (GAS)?

GAS Oferuje:

- automatyczne buffy/debuffy,
- Tagi i replikację,
- Predykcję w multiplayerze,
- Silne oddzielenie danych od kodu,

Dziękuję za uwagę!