

UNIVERSITATEA NAȚIONALĂ DE ȘTIINȚĂ ȘI TEHNOLOGIE POLITEHNICA BUCUREȘTI
CENTRUL UNIVERSITAR PITEȘTI
FACULTATEA DE ELECTRONICĂ, COMUNICAȚII ȘI CALCULATOARE
DEPARTAMENTUL ELECTRONICĂ, CALCULATOARE ȘI INGINERIE ELECTRICĂ
PROGRAMUL DE STUDII UNIVERSITARE DE LICENȚĂ CALCULATOARE

PROIECT DE DIPLOMĂ

Absolvent
Dumitrache George-Nicolae

Conducător științific
Prof. univ. dr. ing. Anghelescu Petre

UNIVERSITATEA NAȚIONALĂ DE ȘTIINȚĂ ȘI TEHNOLOGIE POLITEHNICA BUCUREȘTI
CENTRUL UNIVERSITAR PITEȘTI
FACULTATEA DE ELECTRONICĂ, COMUNICAȚII ȘI CALCULATOARE
DEPARTAMENTUL ELECTRONICĂ, CALCULATOARE ȘI INGINERIE ELECTRICĂ
PROGRAMUL DE STUDII UNIVERSITARE DE LICENȚĂ CALCULATOARE

PROIECT DE DIPLOMĂ

Joc de acțiune de tip RPG dezvoltat în Unreal Engine

Absolvent
Dumitrache George-Nicolae

Conducător științific
Prof. univ. dr. ing. Anghelușcă Petre

Pitești
Sesiune iunie 2025

Cuprins

1. Introducere	1
1.1. Scopul și motivația proiectului	1
1.2. Obiectivele urmărite.....	1
1.3. Metodologia abordată	2
2. Stadiul actual al cercetării.....	3
2.1. Stadiul proiectului de diplomă	3
2.2. Studiu literaturii de specialitate	3
2.3. Proiecte similare și tehnologii.....	3
2.4. Motoare grafice populare pentru jocuri 3D.....	4
3. Fundamente teoretice.....	5
3.2. Unreal Engine	5
3.3. Sistemul de programare vizuală Blueprint.....	5
3.4. Software-ul de design grafic Krita	5
3.5. Software-ul de modelare 3D Blender.....	6
3.6. Platforma de rigging și animații Mixamo	6
3.7. Integrarea resurselor externe de pe platforma Fab.....	6
4. Proiectarea aplicației	7
4.1. Fereastra principală	7
4.2. Crearea de clase Blueprint	8
4.3. Programarea cu Blueprint-uri.....	9
4.3.1. Fereastra Blueprint.....	9
4.3.2. Moduri de programare Blueprint	10
4.3.3. Noduri fundamentale Blueprint	12
4.4. Interfața grafică Widget Blueprint	14
4.4.1. Fereastra Widget Blueprint	14
4.4.2. Elementele folosite în cadrul WBP	15
4.4.3. Modul Graph	15
4.5. Utilizarea aplicației Krita pentru design 2D	17
4.5.1. Crearea conceptului vizual al personajului principal.....	17
4.5.3. Crearea conceptului vizual de armă al personajului principal	18
4.5.4. Alte concepte vizuale de design 2D	19
4.6. Utilizarea aplicației Blender pentru modelare 3D.....	20

4.6.1. Modelarea personajului principal.....	20
4.6.2. Modele 3D de tip recuzită (Props)	23
4.7. Utilizarea platformei Mixamo pentru rigging și animații de caractere.....	25
4.8. Sistemul de animații în Unreal Engine	26
4.8.1. Riguirea personajului principal	26
4.8.2. Ce este un Animation Blueprint	27
4.8.3. Crearea și utilizarea unui State Machine.....	29
4.8.4. Animații de tip Blend Space.....	30
4.8.5. Slot-ul de animație Default Slot.....	30
4.8.6. Montaje de animații (Animation Montage)	31
4.8.7. Condiții de selectare State Machine.....	31
4.8.8. Blueprint-ul de Animatie al personajului principal.....	32
4.9. Utilizarea Sistemului AI din Unreal Engine	34
Componentele principale ale AI în Unreal.....	34
4.10. Landscape.....	34
Caracteristici principale ale Landscape.....	34
5. Implementarea aplicației	35
5.1. Sistemul de control al jucătorului	35
5.1.1. Sub-sistemul de control.....	37
5.1.2. Interacțiuni	44
5.2. Sub-sistemul de luptă	52
5.2.1. Atacuri contextuale	54
5.2.2. Lock-on	60
5.2.3. Trasări de linii pentru atacuri	64
5.2.4. Daune Jucător-Oponent.....	69
5.3. Sub-sistemul de atrbute	72
5.3.1. Viață, rezistență fizică și experiență jucătorului	72
5.3.2. Recuperarea de atrbute ale jucătorului	81
5.4. Sub-sisteme suport	83
5.4.1. Consumabile.....	83
5.4.2. Recompense	84
5.4.3. Harta minimizată a jucătorului (Mini-Map).....	85
5.5. Sistemul de alcătuirea al Punctului de Control (Checkpoint)	87
5.5.1. Sistemul de vreme.....	88
5.5.2. Sub-sistemul de anotimpuri	97

5.5.3. Sub-sistemul de fenomene meteorologice	104
5.5.4. Sistemul de îmbunătățiri al atributelor.....	114
5.5.5. Sistemul de schimbare al înfățișării	118
5.6. Sistemul de control AI.....	123
5.6.1. Prezentarea actorului oponent.....	123
5.6.3. Sub-sistemul de luptă	124
5.6.4. Componentele de control ale sistemului AI	130
5.7. Sistemul de misiuni și NPC-uri pasive	142
5.7.1. Managerul de misiuni.....	142
5.7.2. Misiuni Principale	144
5.7.3. Misiuni Secundare.....	146
5.7.4 NPC-uri pasive	148
5.8. Harta de joc	149
6. Testarea și rezultate.....	150
6.1. Teste de performanță	150
Tabel de performanță	150
Alte dispozitive de test ale performanței.....	151
6.2. Probleme întâmpinate și soluții.....	153
Implementarea blending-ului pentru textura Landscape-ului	153
Probleme de detecție multiplă a coliziunii pentru atacurile inamicilor	153
Calculul greșit al experienței (XP).....	153
7. Contribuții proprii	154
8. Concluzii	155
Bibliografie	156

Figuri

Figură 4.1 – Fereastra principală de editor 3D	7
Figură 4.2 – Meniul de creare asset-uri.....	8
Figură 4.3 – Meniul de selectare a clasei părinte din Blueprint Class.....	8
Figură 4.4 – Fereastra Blueprint	9
Figură 4.5 – Exemplu de nod Blueprint	10
Figură 4.6 – Exemplu de nod Blueprint folosind Custom Event	10
Figură 4.7 – Exemplu de nod Blueprint folosind BPI (1).....	10
Figură 4.8 – Exemplu de nod Blueprint folosind BPI (2).....	10
Figură 4.9 – Lista de funcții BPI.....	11
Figură 4.10 – Panoul de detalii funcții BPI.....	11
Figură 4.11 – Butonul de implementare clase	11
Figură 4.12 – Panoul de detalii al claselor.....	11
Figură 4.13 – Noduri fundamentale de event	12
Figură 4.14 – Noduri fundamentale matematice.....	12
Figură 4.15 – Noduri fundamentale de mișcare	12
Figură 4.16 – Noduri fundamentale de UI	12
Figură 4.17 – Noduri fundamentale de timp	13
Figură 4.18 – Noduri fundamentade pentru controlul AI	13
Figură 4.19 – Noduri fundamentale de obținere a referințelor	13
Figură 4.20 – Fereastra Widget Blueprint.....	14
Figură 4.21 – Componenta text	15
Figură 4.22 – Componenta buton.....	15
Figură 4.23 – Componenta slider.....	15
Figură 4.24 – Componenta imagine	15
Figură 4.25 – Componenta bară de progres	15
Figură 4.26 – Setarea de eveniment onClick pentru buton	16
Figură 4.27 – Dispatcher onClick	16
Figură 4.28 – Setarea de eveniment onValueChanged pentru slider.....	16
Figură 4.29 – Dispatcher onValueChanged.....	16
Figură 4.30 - Fereastra de desen & Designul final al personajului principal	17
Figură 4.31 – Schița personajului principal	17
Figură 4.32 – Conturul personajului principal.....	17
Figură 4.33 – Procesul de concept & design al personajului principal.....	18
Figură 4.34 –Procesul de concept & design a armei personajului principal.....	18
Figură 4.35 – Procesul de concept & design a armei oponenților AI mele	19
Figură 4.36 – Procesul de concept & design a scutului oponenților AI mele	19
Figură 4.37 – Procesul de concept & design a arcului oponenților AI ranged.....	19
Figură 4.38 – Referința model.....	20
Figură 4.39 – Poziționarea referințelor	20
Figură 4.40 – Modelarea 3D a personajului principal.....	20
Figură 4.41 – Subdivizarea modelului	21

Figură 4.42 – Modelarea pieselor de armură.....	21
Figură 4.43 – Modelarea structurii faciale	21
Figură 4.44 – Separarea UV-urilor de model	22
Figură 4.45 - Rezultatul final al modelului 3D	22
Figură 4.46 – Modelul 3D de armă al personajului principal	23
Figură 4.47 – Modele 3D de arme ale oponenților AI, Mele & Ranged.....	23
Figură 4.48 – Modele de iarbă, cuferi, garduri, scări, stânci.....	24
Figură 4.49 – Captură de ecran preluată de pe platforma Mixamo, utilizată cu scop educațional... Figură 4.50 – Modele puse la dispoziție de platforma Mixamo.....	25
Figură 4.51 – Resursele scheletului de model	25
Figură 4.52 – Fereastra de rig și animații a personajului principal.....	26
Figură 4.53 – Fereastra AnimGraph	27
Figură 4.54 – Fereastra EventGraph.....	28
Figură 4.55 – Nodul de update al variabilelor de animație	28
Figură 4.56 – Exemplu State Machine.....	29
Figură 4.57 – Stările unui State Machine	29
Figură 4.58 – Animația unei stări.....	29
Figură 4.59 – Tranzitiiile stărilor	29
Figură 4.60 – Exemplu de tranzitie între stări	29
Figură 4.61 – Animație Blend Space folosită pentru sistemul de Lock-On	30
Figură 4.62 – Nodul Default Slot.....	30
Figură 4.63 – Exemplu de nod Animation Montage.....	31
Figură 4.64 – Nodul de Blend Pose.....	31
Figură 4.65 – AB a personajului principal	32
Figură 4.66 – Organograma AB a personajului principal	32
Figură 4.67 – Stările mașină ale personajului principal	32
Figură 4.68 – Clasa de atribuire AB a personajului principal.....	33
Figură 4.69 – Selectarea modului Landscape	34
Figură 4.70 – Foliage iarbă	34
Figură 5.1 – Organograma generală a sistemului de control al jucătorului	36
Figură 5.2 – Componentele de alcătuire ale actorului jucător	36
Figură 5.3 – Controlul direcției de deplasare (Blueprint)	37
Figură 5.4 – Axele de deplasare ale jucătorului XY	38
Figură 5.5 – Reprezentare vizuală a axeelor de deplasare ale jucătorului XY.....	38
Figură 5.6 – Controlul orientării camerei (Blueprint)	39
Figură 5.7 – Funcționalitatea de săritură Blueprint	40
Figură 5.8 – Funcționalitatea de alergare (Blueprint)	41
Figură 5.9 – Brațul de cameră	42
Figură 5.10 – Reprezentarea vizuală a vitezei de cameră cu lag.....	42
Figură 5.11 – Funcționalitatea de evitare (Blueprint)	43
Figură 5.12 – Reprezentarea vizuală a coliziunilor cu obiecte	44

Figură 5.13 – Componentele de alcătuire a actorului cufăr	45
Figură 5.14 – Reprezentarea vizuală a actorului cufăr	45
Figură 5.15 – Detectarea coliziunilor de cufăr (Blueprint)	46
Figură 5.16 – Oprirea coliziunilor de cufăr (Blueprint)	46
Figură 5.17 – Interacțiunea cu actorul cufăr (Blueprint)	47
Figură 5.18 – Reprezentarea vizuală a interacțiunii cu actorul cufăr	48
Figură 5.19 – Componentele de alcătuire ale actorului ușă	48
Figură 5.20 – Reprezentarea vizuală a interacțiunii cu actorul ușă	49
Figură 5.21 – Interacțiunea cu actorul ușă (Blueprint)	49
Figură 5.22 – Nodul Flip-Flop	49
Figură 5.23 – Reprezentarea vizuală a interacțiunii cu actorul ușă	50
Figură 5.24 – Reprezentarea vizuală a interacțiunii cu NPC	51
Figură 5.25 – Funcționalitatea de echipare și dezechipare a armei	52
Figură 5.26 – Reprezentarea vizuală de echipare și dezechipare a armei	53
Figură 5.27 – Organograma atacurilor contextuale ușoare	54
Figură 5.28 – Atacul normal/multiplu.....	55
Figură 5.29 – Marcarea punctelor în secvența animațiilor de monaj	55
Figură 5.30 – Atacul din alergare (Blueprint).....	56
Figură 5.31 – Atacul din săritură (Blueprint)	56
Figură 5.32 – Organograma atacului greu	58
Figură 5.33 – Atacul greu (Blueprint)	58
Figură 5.34 – Organograma generală Lock-on.....	60
Figură 5.35 – Funcționalitatea Lock-on (Blueprint)	61
Figură 5.36 – Trasarea sferei de detectare a unui actor	61
Figură 5.37 – Fixare cursor Lock-on pe actor (Blueprint)	62
Figură 5.38 – Reprezentarea vizuală a modului Lock-on	62
Figură 5.39 – Eliminarea cursorului de pe ecran (Blueprint)	62
Figură 5.40 – Actualizarea camerei după oponent Lock-on (Blueprint).....	63
Figură 5.41 – Reprezentarea vizuală a punctelor de început și sfârșit trasare linie	64
Figură 5.42 – Componentele de alcătuire a punctelor de începu și sfârșit trasare linie	64
Figură 5.43 – Organograma generală a trasării de linie pentru atacuri	65
Figură 5.44 – Marcarea notificărilor în animația de montaj pentru atac normal/multiplu	65
Figură 5.45 – Marcarea notificărilor în animația de montaj pentru atac greu.....	65
Figură 5.46 – Primirea de notificare (Blueprint).....	66
Figură 5.47 – Sfârșitul de notificare (Blueprint)	66
Figură 5.48 – Modul de comunicarea între BPISwordTrace și jucător	66
Figură 5.49 – Trasarea de linii (Blueprint)	67
Figură 5.50 – Reprezentarea vizuală a trasărilor de linii (1)	68
Figură 5.51 – Reprezentarea vizuală a trasărilor de linii (2)	68
Figură 5.52 – Nodurile de transmitere și primire daune	69
Figură 5.53 – Transmiterea daunelor de către jucător asupra oponentilor (Blueprint)	69
Figură 5.54 – Reprezentarea vizuală a coliziunilor de trasare linii.....	70
Figură 5.55 – Recepționarea daunelor de către jucător (Blueprint)	71

Figură 5.56 – Funcția de reducere a vieții (Blueprint)	72
Figură 5.57 – Funcția de creștere a vieții (Blueprint)	73
Figură 5.58 – Funcția de creștere a vieții maxime (Blueprint)	73
Figură 5.59 – Funcția de reducere a rezistenței fizice (Blueprint).....	74
Figură 5.60 – Funcția de creștere a rezistenței fizice (Blueprint).....	75
Figură 5.61 – Funcția de creștere a rezistenței fizice maxime (Blueprint).....	76
Figură 5.62 – Bara de viață și rezistență fizică	76
Figură 5.63 – Organograma generală a atributului de experiență	77
Figură 5.64 – Funcția de creștere a experienței (Blueprint) (1)	78
Figură 5.65 – Funcția de creștere a experienței (Blueprint) (2)	78
Figură 5.66 – Funcția de creștere a experienței (Blueprint) (3)	78
Figură 5.67 – Funcția de creștere a experienței maxime (Blueprint)	79
Figură 5.68 – Funcția de primire experiență (Blueprint)	80
Figură 5.69 – Bara de experiență	80
Figură 5.70 – Organograma generală a recuperării de atribute ale jucătorului	81
Figură 5.71 – Recuperare de atribute ale jucătorului (Blueprint) (1)	81
Figură 5.72 – Recuperare de atribute ale jucătorului (Blueprint) (2).....	82
Figură 5.73 – Poțiunea de viață	83
Figură 5.74 – Poțiunea de viață (Blueprint)	83
Figură 5.75 – Funcția de recompense din cufere (Blueprint)	84
Figură 5.76 – Funcția de repompense (Blueprint).....	84
Figură 5.77 – Reprezentare vizuală mini-hartă.....	85
Figură 5.78 – Componentele de alcătuire ale mini-hărții	85
Figură 5.79 – Camera mini-hartă	85
Figură 5.80 – Cursorul camerei de mini-hartă.....	86
Figură 5.81 – Interfața jucătorului	86
Figură 5.82 – Organograma generală a punctului de control	87
Figură 5.83 – Interfața Widget UI a punctului de control	88
Figură 5.84 – Componentele de alcătuire ale actorului cer	89
Figură 5.85 – Rotația surselor de lumină (Blueprint)	89
Figură 5.86 – Reprezentarea vizuală a rotației surselor de lumină	91
Figură 5.87 – Gestionarea orelor, minutelor și a secundelor (Blueprint).....	91
Figură 5.88 – Marcarea momentului de timp zi/noapte (Blueprint)	93
Figură 5.89 – Bind-uri buton la apăsare pentru schimbul de timp în joc	94
Figură 5.90 – Funcțiile de setare zi/noapte (Blueprint)	95
Figură 5.91 – Interfața de comunicare pentru setare zi/noapte (Blueprint)	95
Figură 5.92 – Reprezentarea vizuală a scenei pe timp de zi.....	96
Figură 5.93 – Reprezentarea vizuală a scenei pe timp de noapte	96
Figură 5.94 – Lista funcțiilor de funcții ale materialelor	97
Figură 5.95 – Determinarea culorii unui material funcție (Blueprint)	97
Figură 5.96 – Determinarea culorii unui material funcție (Blueprint)	98
Figură 5.97 – Schema bloc a materialului automat a texturii de iarba	99
Figură 5.98 – Nodul de blend al texturilor.....	99

Figură 5.99 – Nodul de separare parametrii	100
Figură 5.100 – Material Parameter Collection	100
Figură 5.101 – Nodul Learp.....	100
Figură 5.102 – Butoanele pentru schimbul de anotimpuri	101
Figură 5.103 – Funcțiile pentru anotimpuri (Blueprint)	101
Figură 5.104 – Setarea materialului landscape	102
Figură 5.105 – Rerezentarea vizuală a anotimpului de primăvară	103
Figură 5.106 – Rerezentarea vizuală a anotimpului de vară.....	103
Figură 5.107 – Rerezentarea vizuală a anotimpului de toamnă	103
Figură 5.108 – Rerezentarea vizuală a anotimpului de iarnă.....	103
Figură 5.109 – Fountain Niagara Emitter	104
Figură 5.110 – Fountain Niagara Emitter Parametrii	104
Figură 5.111 – Fountain Niagara Emitter Parametrii	105
Figură 5.112 – Funcțiile de setare a vremii (Blueprint).....	105
Figură 5.113 – Activarea și dezactivarea fenomenelor (Blueprint)	106
Figură 5.114 – Reprezentarea vizuală a fenomenelui de ploaie	107
Figură 5.115 – Reprezentarea vizuală a fenomenelui de ninsoare.....	107
Figură 5.116 – Butonul de ceată	108
Figură 5.117 – Resetarea de ceată (Blueprint).....	108
Figură 5.118 – Funcția managerului de vreme (Blueprint)	109
Figură 5.119 – Organograma generală a managerului de vreme.....	109
Figură 5.120 – Exemplu funcție de ambientă a scenei	111
Figură 5.121 – Reprezentarea vizuală a fenomenului de ceată pe timp de zi	112
Figură 5.122 – Reprezentarea vizuală a fenomenului de ceată pe timp de noapte	112
Figură 5.123 – Butoanele pentru îmbunătățire atribute și resetare puncte.....	114
Figură 5.124 – Organograma generală a îmbunătățirii de atribute	114
Figură 5.125 – Funcția de îmbunătățire a vietii (Blueprint)	115
Figură 5.126 – Organograma generală a resetării de atribute.....	116
Figură 5.127 – Funcția de resetare atribute (Blueprint).....	116
Figură 5.128 – Funcția de actualizare statistici (Blueprint)	117
Figură 5.129 – Slidere RGB și preview de culoare	118
Figură 5.130 – Slidere pentru setarea componentei Red (Blueprint)	119
Figură 5.131 – Slidere pentru setarea componentei Green (Blueprint)	119
Figură 5.132 – Slidere pentru setarea componentei Blue (Blueprint)	119
Figură 5.133 – Slidere pentru setarea componentei Blue (Blueprint)	120
Figură 5.134 – Setarea parametrilor RGB în MPC	120
Figură 5.135 – Slidere pentru setarea componentei Blue (Blueprint)	120
Figură 5.136 – Reprezentarea vizuală a schimbului de infățișare	121
Figură 5.137 – Actualizarea de infățișare (Blueprint)	121
Figură 5.138 – Organograma generală de actualizarea de infățișării	122
Figură 5.139 – Actualizarea de infățișare (Blueprint)	122
Figură 5.140 – Componentele de alcătuire ale actorului oponent	123
Figură 5.141 – Variante de actori oponent	123

Figură 5.142 – Bara de viață și nivel	124
Figură 5.143 – Rotația bării de viață și nivel	124
Figură 5.144 – Transmiterea daunelor de către oponent mele asupra jucătorului (Blueprint)	125
Figură 5.145 – Evitarea de daune (Blueprint)	125
Figură 5.146 – Transmiterea daunelor de către oponent ranged asupra jucătorului (Blueprint)	125
Figură 5.147 – Recepția daunelor de către oponent (Blueprint) (1)	126
Figură 5.148 – Recepția daunelor de către oponent (Blueprint) (2)	126
Figură 5.149 – Recepția daunelor de către oponent (Blueprint) (3)	127
Figură 5.150 – Setarea de viață și daune în funcție de nivel (Blueprint) (2)	128
Figură 5.151 – Setarea de viață și daune în funcție de nivel (Blueprint) (1)	128
Figură 5.152 – Componentele controller-ului AI	130
Figură 5.153 – Detectarea jucătorului de către AI (Blueprint)	130
Figură 5.154 – Blackboard-ul AI	131
Figură 5.155 – Behavior Tree.....	131
Figură 5.156 – Stările AI (Behavior Tree)	132
Figură 5.157 – Organograma generală a controlului AI.....	132
Figură 5.158 – Starea de patrulare a AI-ului	133
Figură 5.159 – Task-ul de rută de patrulare	133
Figură 5.160 – Task-ul de urmare a rutei.....	133
Figură 5.161 – Funcția de obținere a punctelor a liniei de patrulă	134
Figură 5.162 – Reprezentarea vizuală a linie de patrulă.....	134
Figură 5.163 – Blueprint a liniei de patrulă	134
Figură 5.164 – Starea de echipare armă a AI-ului	135
Figură 5.165 – Condiția de echipare a armei AI	135
Figură 5.166 – Task-ul de echipare a armei AI	135
Figură 5.167 – Funcția de echipare armă AI	135
Figură 5.168 – Starea de urmărire și atac a AI-ului	136
Figură 5.169 – Task-ul de ștergere focusare AI asupra jucătorului.....	136
Figură 5.170 – Task-ul de deplasare AI spre jucătorului	136
Figură 5.171 – Task-ul de focusare AI asupra jucătorului	136
Figură 5.172 – Task-ul de atac AI	137
Figură 5.173 – Funcția de atac AI	137
Figură 5.174 – Stare de pierdere din vedere a jucătorului	137
Figură 5.175 – Task-ul de dezechipare armă AI	137
Figură 5.176 – Funcția dedezechipare armă AI	138
Figură 5.177 – Task-ul de resetare poziție și rotație la start	138
Figură 5.178 – Funcția de resetare poziție și rotație la start	138
Figură 5.179 – Funcția de resetare poziție și rotație la start	139
Figură 5.180 – Starea de strafe	139
Figură 5.181 – Task-ul de strafe AI	139
Figură 5.182 – EQS.....	140
Figură 5.183 – Reprezentarea vizuală de calcul al pathfinding-ului.....	140
Figură 5.184 – Funcționalitatea distance to query.....	141

Figură 5.185 – Reprezentarea vizuală a EQS de strafe	141
Figură 5.186 – Funcționalitatea OnCircle	141
Figură 5.187 – Widgetul lista de misiuni.....	142
Figură 5.188 – Inițializarea variabilelor pentru misiuni (Blueprint).....	143
Figură 5.189 – Selectarea de misiuni principale (Blueprint).....	144
Figură 5.190 – Adăugarea de misiuni principale (Blueprint)	144
Figură 5.191 – Structura de text folosită în cadrul misiunilor	144
Figură 5.192 – Updatarea de misiuni principale (Blueprint).....	145
Figură 5.193 – Incrementarea numărului misiunii principale (Blueprint)	145
Figură 5.194 – Ștergerea de misiună principale (Blueprint)	145
Figură 5.195 – Parcugerea array-ului de indexi ai text box-urilor (Blueprint)	146
Figură 5.196 – Parcugerea array-ului de referințe ale text box-urilor (Blueprint).....	146
Figură 5.197 – Ocuparea indexilor și atribuirea de misiune (Blueprint)	146
Figură 5.198 – Parcugem array-ul de ID-uri (Blueprint).....	147
Figură 5.199 – Updatăm textul misiunii găsite(Blueprint)	147
Figură 5.200 – Parcugere array ID-uri pentru misiunea completată (Blueprint).....	147
Figură 5.201 – Stergem misiunea din array-uri (Blueprint)	148
Figură 5.202 – NPC-urile utilizate (Blueprint)	148
Figură 5.203 – Interacțiunea cu NPC-uri (Blueprint)	148
Figură 5.204 – Harta de joc	149
Figură 6.1 – Setările de scalabilitate ale calității video	150

Tabele

Tabel 5.1 – Tabelul de interpretare al valorilor pentru determinarea direcției de deplasare	38
Tabel 5.2 – Tabelul funcțiilor de orientare ale camerei	39
Tabel 5.3 – Tabelul atacurilor ușoare	55
Tabel 5.4 – Tabelul atacurilor contextuale	59
Tabel 5.5 – Tabelul rotației surselor de lumină	90
Tabel 5.6 – Tabelul combinațiilor pentru determinarea de anotimp.....	102
Tabel 5.7 – Tabelul combinațiilor pentru determinarea de fenomene meteorologice (ploaie/ninsoare)	106
Tabel 5.8 – Tabelul combinațiilor de anotimpuri, momente ale zilei și fenomene meteorologice..	113
Tabel 5.9 – Tabelul de reprezentare al statisticilor oponenților în funcție de nivel.....	129
Tabel 6.1 – Tabelul de performanță (1)	150
Tabel 6.2 – Tabelul de performanță (2)	152
Tabel 6.3 – Tabelul de performanță (3)	152

Lista acronime

- ABP / AB** – Animation Blueprint (Blueprint de animație)
- AI** – Artificial Intelligence (Inteligentă Artificială)
- AAA** – Triple-A (Jocuri video cu bugete mari de producție)
- BB** – Blackboard (Tabel de variabile pentru AI Behavior Tree)
- BPI** – Blueprint Interface (Interfață Blueprint)
- BP** – Blueprint (Sistem de script vizual)
- BT** – Behavior Tree (Arbore de comportament pentru AI)
- EQS** – Environmental Query System (Sistem de interogare a mediului)
- FOV** – Field of View (Câmp vizual)
- FPS** – Frames Per Second (Cadre pe secundă)
- HUD** – Heads-Up Display (Interfață grafică pe ecran)
- ID** – Identifier (Identifier)
- IK** – Inverse Kinematics (Cinematica inversă – pentru animații)
- LOD** – Level of Detail (Nivel de detaliu grafic)
- NPC** – Non-Playable Character (Personaj Non-jucabil)
- PBR** – Physically Based Rendering (Randare bazată pe fizică – materiale realiste)
- PC** – Personal Computer (Calculator personal)
- RPG** – Role Playing Game (Joc de rol)
- RGB** – Red Green Blue (Componente de culoare)
- RPC** – Remote Procedure Call (Apel de procedură la distanță – rețea)
- RM** – Mișcare bazată pe animație (Root Motion)
- SM** – Static Mesh (Rețea statică / model 3D static)
- UMG** – Unreal Motion Graphics (Sistem UI Unreal)
- WBP** – Widget Blueprint (Interfețe UI)

1. Introducere

1.1. Scopul și motivația proiectului

Scopul acestui proiect de diplomă a fost realizarea unui joc de acțiune 3D de tip Fantasy RPG utilizând motorul grafic Unreal Engine. Acest gen de joc presupune o lume imaginară, complexă și interactivă, în care jucătorul își asumă rolul unui personaj având posibilitatea de a explora, experimenta și depăși diverse provocări pentru a avansa în poveste.

Motivația alegerii acestui subiect provine din pasiunea profundă pentru dezvoltarea jocurilor video, un domeniu în continuă expansiune și cu un impact semnificativ în industria divertismentului digital. De asemenea, am dorit să aplic într-un mod practic cunoștințele dobândite în timpul studiilor universitare, precum programarea logică, modelarea conceptelor prin diagrame de stare și flux, precum și noțiuni de modelare 3D și grafică pe calculator.

Proiectul contribuie nu doar la aplicare cunoștințelor tehnice dobândite în cadrul facultății, ci și la dezvoltarea competențelor practice necesare în domeniul dezvoltării de jocuri video, precum gădirea algoritmică, design-ul sistemic și integrarea elementelor vizuale și interactive.

1.2. Obiectivele urmărite

Obiectivele urmărite în cadrul acestui proiectului vizează dezvoltarea unei experiențe interactive și captivante prin integrarea unor mecanici complexe specifice genului RPG, care să asigure atât funcționalitatea, cât și imersiunea în lumea virtuală creată prin implementarea următoarelor sisteme:

- Sistemul de control al personajului
- Sistemul de luptă al personajului
- Sistemul de gestionare a atributelor personajului
- Sistemul de experiență și nivel
- Sistemul de simulare a timpului, anotimpurilor și condițiilor meteorologice
- Sistemul de îmbunătățire a atributelor
- Sistemul de schimbare a infățișării personajului
- Sistemul de control AI a oponentilor
- Sistemul de misiuni și progres al jocului
- Sistemul de NPC-uri pasive
- Harta interactivă din joc

Aceste obiective au fost structurate pentru a reflecta complexitatea unui proiect de joc complet funcțional, urmărind coerentă între gameplay, elemente vizuale și interacțiuni.

1.3. Metodologia abordată

Pentru realizarea proiectului, a fost adoptată o metodologie practică, centrată pe utilizarea motorului grafic Unreal Engine, care oferă un mediu robust și versatil pentru dezvoltarea de jocuri video 3D. Procesul de dezvoltare a fost împărțit în mai multe etape, fiecare având obiective specifice și contribuind la construcția treptată a jocului:

1. **Documentarea și analiza genului RPG** – a fost realizată o analiză a caracteristicilor definitorii ale jocurilor RPG, identificând elementele esențiale.
2. **Planificarea și schițarea structurii jocului** – au fost stabilite funcționalitățile principale, structura narrativă de bază, precum și arhitectura sistemelor necesare. În această etapă s-au folosit diagrame de flux și scheme logice pentru a contura comportamentul fiecărui sistem.
3. **Dezvoltarea componentelor modulare** – jocul a fost construit modular, fiecare sub-sistem fiind dezvoltat și testat separat în sistemul vizual de programare Blueprint oferit de Unreal Engine.
4. **Integrarea sistemelor și testarea** – după dezvoltarea fiecărui modul, s-a realizat integrarea treptată a acestora într-un proiect unitar. Au fost realizate teste funcționale pentru a verifica compatibilitatea și stabilitatea fiecărui sistem în ansamblul jocului.
5. **Optimizare și ajustări finale** – ultima etapă a vizat îmbunătățirea performanței generale a jocului, ajustarea detaliilor grafice, echilibrarea gameplay-ului și rafinarea interfeței utilizatorului pentru o experiență cât mai intuitivă.

2. Stadiul actual al cercetării

2.1. Stadiul proiectului de diplomă

Jocul dezvoltat reprezintă o bază solidă și funcțională, ce oferă un mediu minimalist de explorare, însotit de elemente esențiale precum misiuni, oponenți și secrete. Deși complexitatea jocului este momentan redusă, structura sa modulară și mecanicile implementate asigură un fundament puternic pentru dezvoltări și îmbunătățiri viitoare. Astfel, proiectul poate fi extins prin adăugarea de noi sisteme, conținut și caracteristici, în vederea realizării unei experiențe RPG mai bogate și mai captivante.

2.2. Studiul literaturii de specialitate

Industria jocurilor video reprezintă una dintre cele mai dinamice și influente ramuri ale divertismentului digital. Genul RPG a evoluat semnificativ în ultimele decenii, trecând de la jocuri 2D cu mecanici simple la experiențe 3D complexe și profund imersive. Aceste jocuri se caracterizează prin sisteme de progresie a personajelor, narături ramificate, lumi vaste și interactive, precum și o interacțiune complexă între jucător și mediu.

Un element esențial în dezvoltarea RPG-urilor moderne este integrarea unor componente avansate, precum inteligența artificială, sisteme personalizate de luptă, medii dinamice (vreme, cicluri/noapte) și interfețe intuitive ce facilitează explorarea și luarea deciziilor. Totodată, tot mai multe jocuri adoptă un stil vizual realist sau stilizat, folosind tehnologii moderne de randare, simulare fizică și animație.

Un **joc RPG** este un tip de joc video în care jucătorul preia controlul unuia sau mai multor personaje într-un univers fictiv. Personajele evoluează prin luarea de decizii, acumularea experienței, dezvoltarea abilităților și interacțiunea cu lumea înconjurătoare. Elementele definitorii ale unui RPG includ sistemul de progresie al personajului, lupte tactice sau în timp real, inventar, misiuni și un fir narativ bine conturat. [1]

Termenul **Fantasy** desemnează un gen narrativ plasat într-un univers imaginar, adesea populat cu creațuri mitologice, magie, regi și tărâmuri neobișnuite. În contextul jocurilor video, genul fantasy oferă un cadru ficitonal ce permite explorarea unor lumi alternative, cu legi proprii, personaje fantastice și aventuri epice, distincte de realitatea cotidiană. [2]

2.3. Proiecte similare și tehnologii

Pe piață actuală există numeroase exemple relevante de jocuri fantasy RPG care au influențat atât designul mecanicilor, cât și direcția artistică a genului. Ca de exemplu:

- **Dark Souls (FromSoftware, 2011)** – este un RPG de acțiune cunoscut pentru dificultatea sa provocatoare și sistemul complex de luptă, care recompensează strategia și răbdarea. Jocul pune accent pe explorarea unui univers întunecat și atmosferic, plin de mister și oponenți redutabili. [3]
- **Baldur's Gate 3 (Larian Studios, 2023)** – este un RPG clasic, bazat pe regulile Dungeons & Dragons, care oferă o poveste captivantă și decizii narrative ce influențează evoluția jocului. Este apreciat pentru modul său profund de dezvoltare a personajelor și interacțiunea cu o lume detaliată. [4]
- **The Witcher 3: Wild Hunt (CD Projekt Red, 2015)** – este un RPG open-world cu o poveste complexă și personaje bine conturate, care oferă o lume vastă, dinamică și plină de misiuni. Jocul combină luptele tactice cu o explorare bogată și un sistem avansat de progresie a personajului. [5]

2.4. Motoare grafice populare pentru jocuri 3D

Motorul grafic reprezintă un ansamblu de instrumente software conceput pentru a facilita dezvoltarea jocurilor video. Acesta furnizează funcționalități esențiale precum: randarea grafică 2D/3D, simularea fizicii și coliziunilor, gestionarea sunetului, animația personajelor, scriptingul (logica jocului), inteligența artificială, precum și suportul pentru interfață și optimizare. [6]

Motoarele grafice permit dezvoltatorilor să creeze jocuri complexe într-un mod eficient, reducând considerabil timpul și resursele necesare pentru a construi aceste funcționalități de la zero.

În prezent, dezvoltatorii de jocuri video beneficiază de o gamă variată de platforme și motoare grafice specializate în crearea de jocuri 3D. Aceste platforme oferă funcționalități avansate, interfețe prietenoase și un ecosistem de instrumente care simplifică procesul de dezvoltare. Cele mai populare platforme sunt:

- **Unreal Engine** este unul dintre cele mai performante motoare grafice disponibile. Este recunoscut pentru capacitatele sale avansate de randare în timp real, grafică fotorealistă și integrarea cu limbajul de programare vizual Blueprint. Este utilizat atât pentru jocuri AAA, cât și pentru proiecte indie. Suportă dezvoltarea pentru PC, console și dispozitive mobile.
- **Unity** este o platformă foarte populară în special în rândul dezvoltatorilor indie și pentru proiecte mobile. Deși este cunoscut inițial pentru suportul 2D, Unity oferă și capabilități solide pentru dezvoltarea de jocuri 3D. Este apreciat pentru flexibilitatea sa, comunitatea largă și integrarea facilă cu limbaje precum C#.
- **Godot** este un motor de jocuri open-source care a crescut semnificativ în popularitate. Oferă suport pentru jocuri 2D și 3D și vine cu un limbaj de scripting propriu (GDScript), precum și suport pentru C#. Este apreciat pentru simplitatea și libertatea oferită dezvoltatorilor.

3. Fundamente teoretice

3.1. Software-uri utilizate

În dezvoltarea au fost utilizate mai multe unelte software specializate, fiecare având un rol esențial în crearea și funcționarea jocului. Alegerea acestor instrumente s-a bazat pe compatibilitatea lor cu motorul de jocuri Unreal Engine, pe eficiență în procesul de prototipare și implementare, precum și pe posibilitățile extinse de personalizare și optimizare.

Pentru dezvoltarea jocului au fost utilizate următoarele unelte software esențiale:

- **Unreal Engine** – motorul principal de joc, care oferă un mediu complet de dezvoltare.
- **Blueprint** – sistemul vizual de programare utilizat pentru prototipare rapidă și implementarea funcționalităților.
- **Krita** – software pentru design grafic 2D, folosit pentru crearea elementelor vizuale.
- **Blender** – aplicație pentru modelare 3D, utilizată în realizarea modelelor.
- **Mixamo** – platformă pentru rigging și animații automate ale personajelor.
- **Fab.com** – sursă online pentru asseturi utilizate în proiect.

3.2. Unreal Engine

Unreal Engine este un motor grafic dezvoltat de Epic Games, recunoscut pe scară largă în industria jocurilor video pentru performanță, flexibilitatea și calitatea vizuală pe care o oferă. Aceasta reprezintă o platformă completă și integrată, care facilitează dezvoltarea rapidă și eficientă a jocurilor video și aplicațiilor interactive 3D complexe. Motorul pune la dispoziție un set bogat de unelte pentru crearea jocurilor 3D, simulărilor, realității virtuale și augmentate, precum și a producțiilor cinematografice, fiind folosit atât de studiouri mari, cât și de dezvoltatori independenți. [7]

3.3. Sistemul de programare vizuală Blueprint

Blueprint este un sistem vizual de scripting integrat în Unreal Engine, care permite crearea logicii jocului fără a scrie cod tradițional. Pe lângă programarea în C++, Blueprint oferă o metodă mult mai intuitivă și accesibilă, ușurând procesul de dezvoltare, în special pentru cei care nu sunt familiarizați cu limbajele de programare clasice. Aceasta facilitează prototiparea rapidă și implementarea funcționalităților complexe prin utilizarea de noduri și conexiuni vizuale, oferind dezvoltatorilor, indiferent de nivelul lor de experiență, un instrument eficient pentru a construi comportamente și interacțiuni în joc. [8]

3.4. Software-ul de design grafic Krita

Krita este un software open-source dedicat designului grafic și picturii digitale, folosit în proiectul nostru pentru crearea și editarea elementelor 2D, cum ar fi conceptele de personaje, modele, texturi, iconițele și alte componente vizuale ale jocului. Krita oferă o gamă largă de pensule și instrumente profesionale, fiind apreciat pentru flexibilitate și ușurință în utilizare. [9]

3.5. Software-ul de modelare 3D Blender

Blender este un software open-source de modelare 3D, utilizat pe scară largă în industrie pentru crearea de modele, animații, efecte vizuale și simulări. În cadrul proiectului de față, Blender a fost folosit pentru modelarea personajelor și obiectelor. [10]

Blender oferă o suită completă de unelte pentru sculptare, texturare, rigging și animare, fiind compatibil cu motoarele de joc precum Unreal Engine. Datorită suportului pentru formate standard (precum .FBX și .OBJ).

3.6. Platforma de rigging și animații Mixamo

Mixamo este o platformă online dezvoltată de Adobe, dedicată rigging-ului automat al modelelor 3D și aplicării de animații predefinite. În cadrul acestui proiect, Mixamo a fost utilizat pentru a genera rapid schelete (rig-uri) și animații pentru personajele 3D modelate în Blender.

Platforma oferă o bibliotecă vastă de animații (mers, alergare, atacuri, interacțiuni etc.) care pot fi aplicate direct pe modele 3D, eliminând necesitatea unui proces complex de animare manuală. Animațiile și rigurile obținute au fost exportate în format compatibil cu Unreal Engine (de obicei .FBX), facilitând integrarea în cadrul blueprint-urilor de animație și a sistemelor de stare (state machines). [11]

Mixamo a contribuit astfel la accelerarea procesului de dezvoltare a personajelor animate, oferind un echilibru între calitate, accesibilitate și eficiență.

3.7. Integrarea resurselor externe de pe platforma Fab

Pentru a accelera procesul de dezvoltare a conținutului vizual și a îmbunătăți calitatea grafică a jocului, au fost utilizate resurse externe disponibile pe platforma Fab.com (anterior cunoscută ca Sketchfab). Aceasta este o bibliotecă digitală ce oferă o gamă largă de modele 3D, texturi, efecte vizuale și alte elemente grafice, majoritatea compatibile cu motoare de joc precum Unreal Engine.

Fab.com pune la dispoziție atât asseturi gratuite, cât și comerciale, realizate de artiști și dezvoltatori din întreaga lume. În cadrul proiectului, au fost selectate modele optimizate pentru performanță și compatibilitate cu Unreal Engine, contribuind la crearea unui mediu de joc coerent și atractiv vizual. [12]

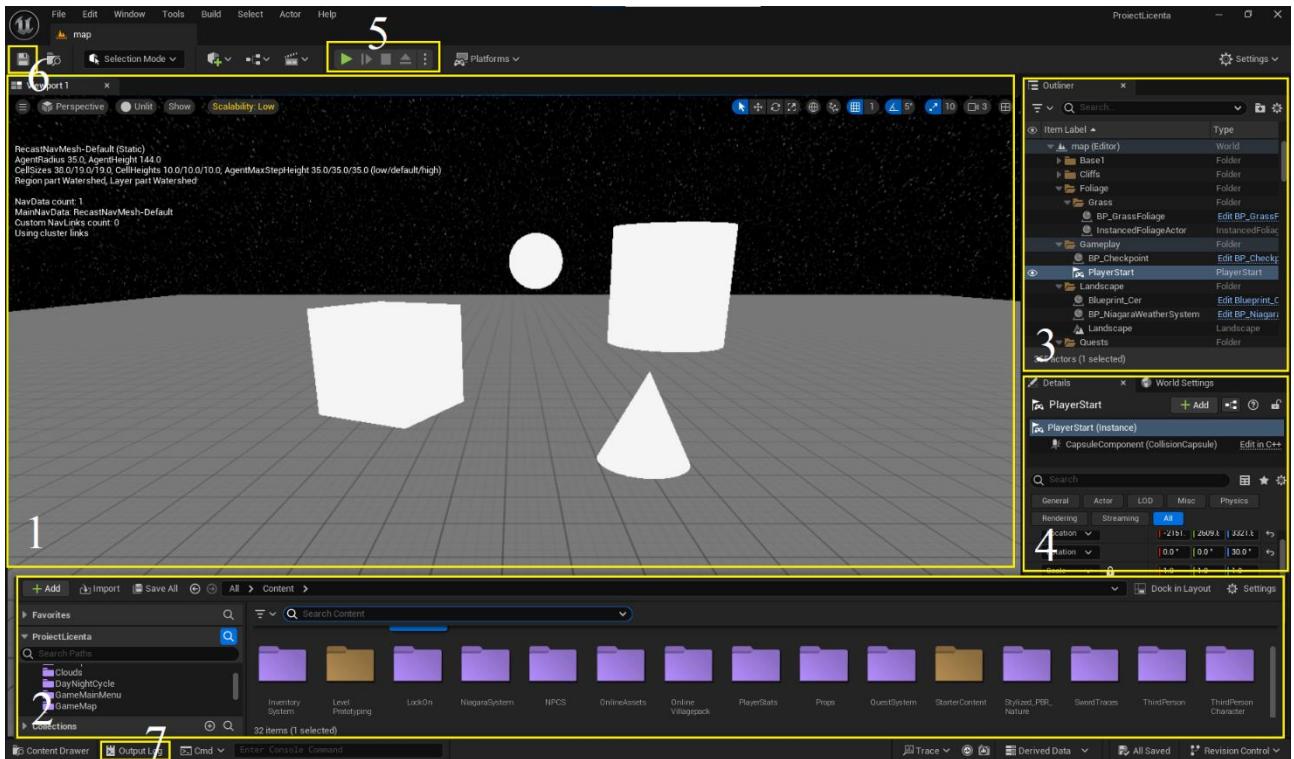
Am avut grija ca resurselor să fie gratis de folosit, cu scop dezvoltării educaționale și nu de a comercializa într-un fel sau altul lucrarea.

Resursele folosite:

- **Stylized Nature Pack [13]**
- **Medieval Castle Asset Pack [14]**

4. Proiectarea aplicației

4.1. Fereastra principală

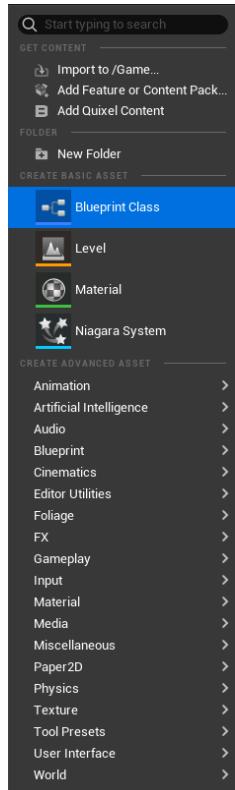


Figură 4.1 – Fereastra principală de editor 3D

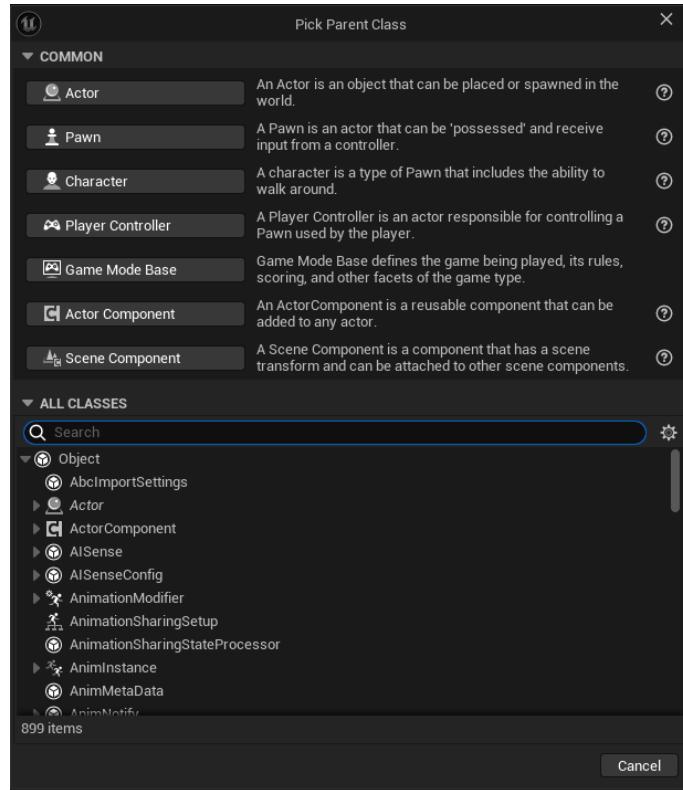
Editorul Unreal Engine oferă o interfață completă și modulară, concepută pentru a facilita editarea, plasarea și organizarea elementelor într-o scenă 3D, conform [Figură 4.1]. Componentele principale ale acestei interfețe sunt:

- 1. Viewport (Fereastra de vizualizare 3D)** – zona principală unde utilizatorul poate plasa, muta și edita obiecte (actori) în cadrul scenei. Aici se desfășoară vizualizarea în timp real a mediului creat.
- 2. Content Drawer (Depozitul de resurse)** – o interfață de tip browser de fișiere unde sunt gestionate asset-urile proiectului: modele 3D, materiale, texturi, sunete, blueprints etc.
- 3. Outliner** – listă ierarhică a tuturor actorilor și obiectelor prezente în scenă. Permite selectarea rapidă a elementelor și organizarea acestora pe categorii sau grupuri.
- 4. Details Panel** – panoul de proprietăți al obiectelor selectate. Aici se pot modifica poziția, rotația, scalarea, precum și alte atrbute specifice (materiale, comportament etc.).
- 5. Bara de simulare (Play / Pause / Stop)** – permite rularea jocului direct în editor pentru a testa gameplay-ul sau comportamentul obiectelor în timp real.
- 6. Salvarea proiectului** – opțiunea de salvare rapidă a nivelului curent sau a întregului proiect, accesibilă din bara de unele sau meniul File.
- 7. Output Log (Consola)** – afișează mesaje generate de motor în timpul rulării proiectului: erori, avertismente, mesaje de sistem sau loguri personalizate. Esențial pentru depanare și analiză în timpul dezvoltării.

4.2. Crearea de clase Blueprint



Figură 4.2 – Meniul de creare asset-uri



Figură 4.3 – Meniul de selectare a clasei părinte din Blueprint Class

În Unreal Engine, meniul de creare asset-uri este fereastra care apare când dăm **click dreapta** în **Content Drawer** și oferă opțiuni pentru a crea diferite tipuri de asset-uri [Figură 4.2], cum ar fi:

- **Blueprint Class** – pentru a crea o nouă clasă Blueprint, adică un şablon pentru actori sau alte obiecte logice.
- **Material** – pentru a crea materiale pentru texturarea obiectelor.
- **Texture** – pentru a crea texturi.
- **Animation** – pentru a crea animații sau blend space-uri.
- **Level** – pentru a crea un nou nivel/hartă.
- **Particle System** – pentru efecte vizuale de particule.
- **Sound** – pentru asset-uri audio.

Și altele.

Pentru dezvoltarea jocului, vom lucra în principal cu **Actori**, **Materiale**, **Niagara System** **Interfețe de Blueprint**, **AI** și **Widget-uri Blueprint (UI)**. [Figură 4.2]

Atunci când creăm un nou Blueprint în Unreal Engine, ni se solicită să alegem o **clasă părinte**. [Figură 4.3] Acest pas este esențial deoarece determină comportamentul de bază și funcționalitățile pe care noul Blueprint le va moșteni. Meniul de selectare a clasei părinte oferă o listă de clase predefinite din motorul Unreal fiecare cu un scop bine definit:

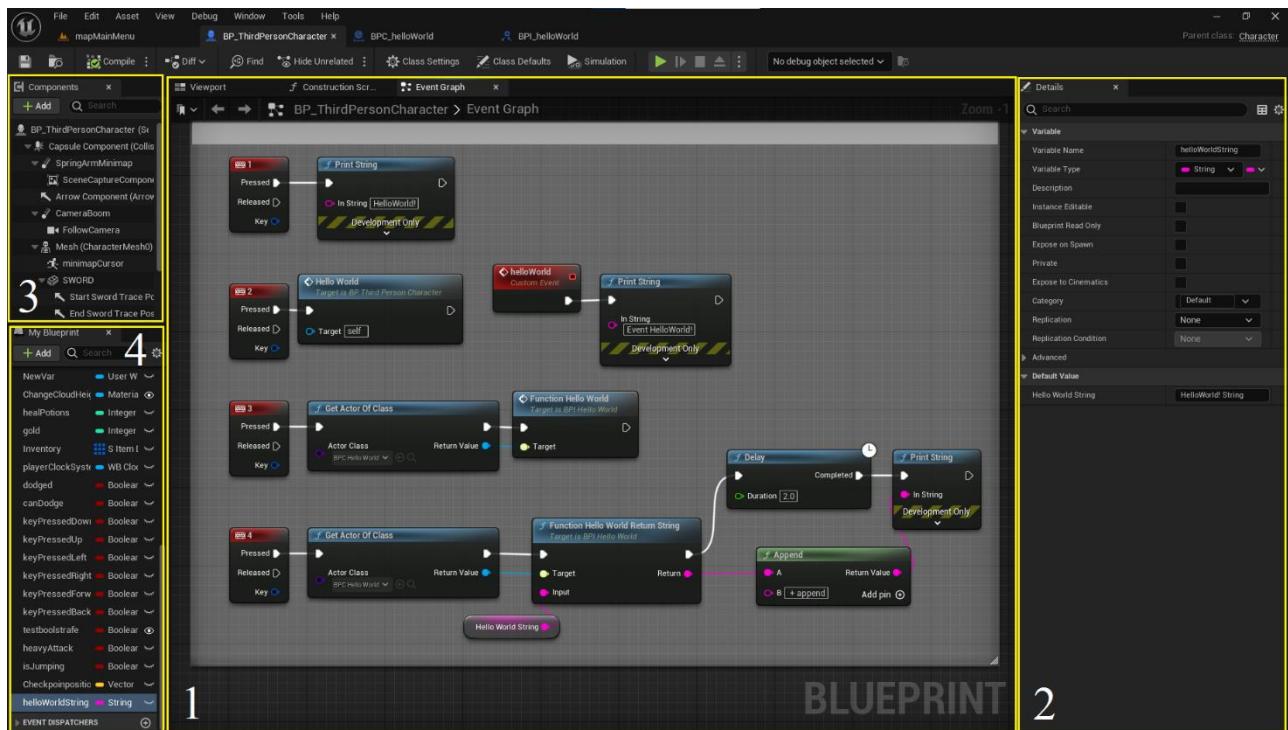
- **Actor** – pentru obiecte generale ce pot fi plasate în scenă.
- **Pawn** – pentru obiecte controlabile de jucător sau AI.
- **Character** – pentru personaje cu mișcare și sistem de animații.
- **Player Controller** – pentru gestionarea input-ului și a interacțiunii cu Pawn-ul.
- **Game Mode Base** – pentru logica de joc (reguli, condiții de victorie etc.).
- **SceneComponent** – pentru logica modulară atașabilă altor actori.

Alegerea corectă a clasei părinte este importantă deoarece determină comportamentul de bază și restricțiile Blueprint-ului creat. De exemplu, dacă dorim să dezvoltăm un obiect interactiv din scenă, vom porni de la **Actor**, în timp ce pentru un personaj controlabil vom selecta **Character**.

4.3. Programarea cu Blueprint-uri

Programarea cu Blueprint-uri reprezintă o metodă vizuală și intuitivă de a crea logica jocului în Unreal Engine, fără a necesita cunoștințe avansate de programare în C++. Sistemul Blueprint utilizează o interfață bazată pe noduri, care permite dezvoltatorilor să construiască și să gestioneze comportamente complexe ale obiectelor, evenimentelor și interacțiunilor din joc. [15]

4.3.1. Fereastra Blueprint



Figură 4.4 – Fereastra Blueprint

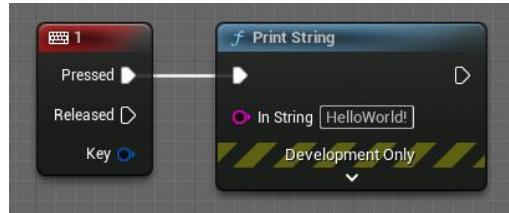
Fereastra Blueprint [Figură 4.4] este compusă din următoarele elemente principale:

- Zona de scripting vizual (Graph Editor)** – permite plasarea și conectarea nodurilor pentru definirea logicii comportamentale a Blueprint-ului (evenimente, funcții, variabile, condiții, cicluri etc.).
- Panoul de detalii (Details Panel)** – afișează proprietățile și atributele elementelor selectate, inclusiv variabile, parametri sau componente, oferind posibilitatea de configurare a acestora.
- Lista de componente (Component Hierarchy)** – prezintă ierarhia componentelor care alcătuiesc actorul definit în Blueprint; aici se pot adăuga, elimina sau organiza componente precum mesh-uri, coliziuni, lumini, camere etc.
- My Blueprint panel** – conține lista completă a elementelor definite în Blueprint: variabile, funcții, evenimente, macro-uri, grafuri, interfețe implementate și timeline-uri.

4.3.2. Moduri de programare Blueprint

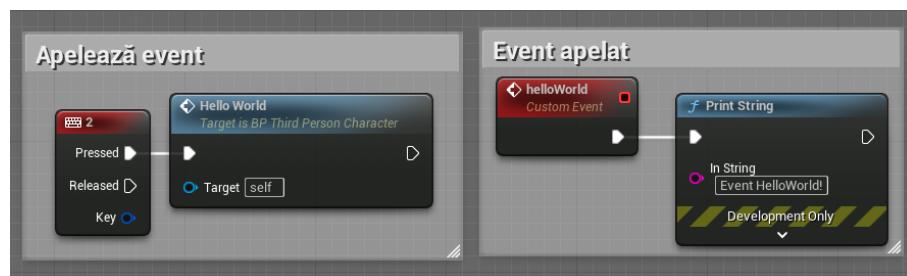
În cadrul proiectului am folosit mai multe concepte esențiale ale programării cu Blueprint-uri, printre care:

- **Logica simplă construită cu ajutorul nodurilor**, pentru a crea comportamente clare și ușor de urmărit. [Figură 4.5]



Figură 4.5 – Exemplu de nod Blueprint

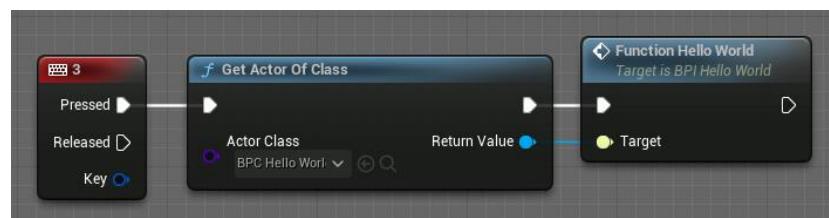
- **Evenimente (Custom Events)** definite în cadrul Blueprint-urilor, care permit reacții dinamice la diverse situații din joc. [Figură 4.6] [16]



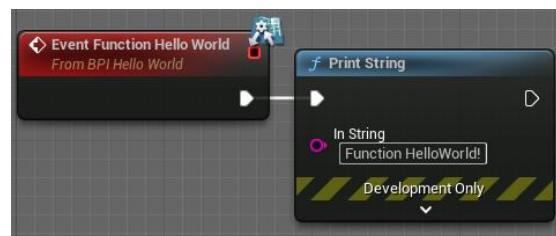
Figură 4.6 – Exemplu de nod Blueprint folosind Custom Event

- **Interfața Blueprint Interfaces (BPI)**, utilizată pentru a facilita comunicarea între două sau mai multe Blueprint-uri.

Exemplu: BP_Jucător [Figura 4.7] către BP_HelloWorld [Figura 4.8] prin intermediul interfeței BPI_HelloWorld cu funcția **functionHelloWorld()**. [Figura 4.9]

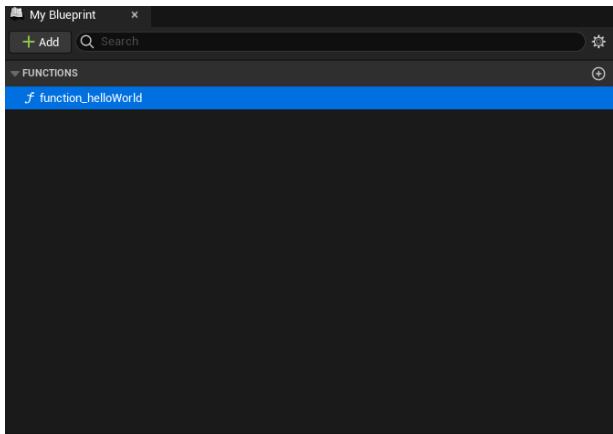


Figură 4.7 – Exemplu de nod Blueprint folosind BPI (1)

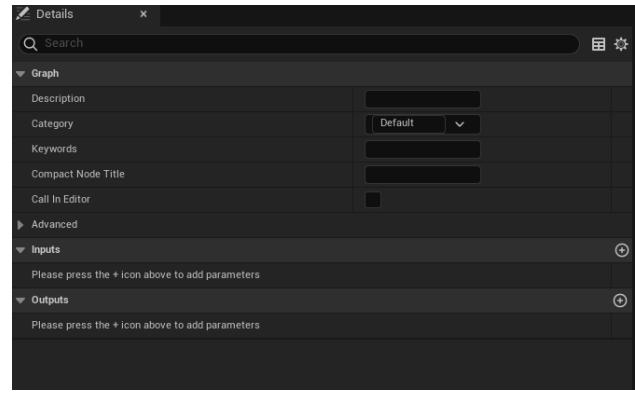


Figură 4.8 – Exemplu de nod Blueprint folosind BPI (2)

Funcțiile BPI pot fi considerate un mecanisme standardizate de comunicare între două sau mai multe Blueprint-uri. Aceste funcții oferă "căi de acces" pentru apelarea unor comportamente sau pentru transmiterea de date între componente distințe ale jocului, fără a crea o dependență directă între clase. Practic, prin intermediul BPI, un Blueprint poate apela funcțiile altuia, trimițând sau primind date prin parametri de intrare/ieșire din detaliile fiecărei funcții. [17]



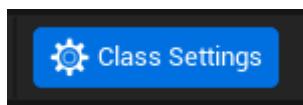
Figură 4.9 – Lista de funcții BPI



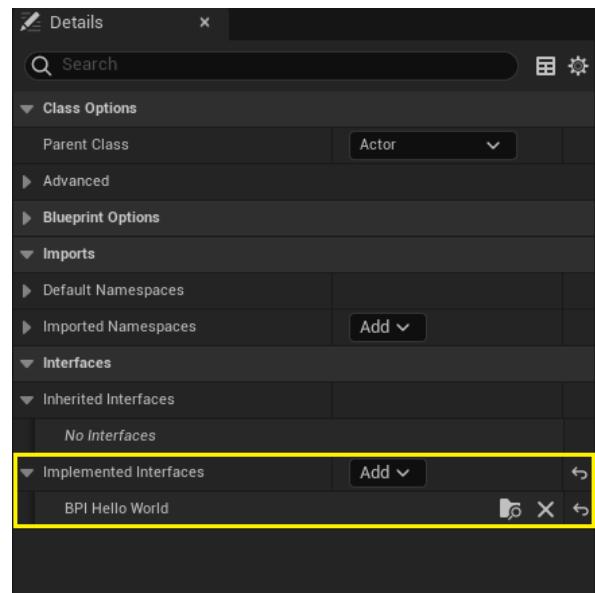
Figură 4.10 – Panoul de detalii funcții BPI

O interfață Blueprint este alcătuită dintr-o listă de funcții abstractive [Figură 4.9] și un panou de detalii [Figură 4.10] unde se configuraază semnătura funcțiilor.

Pentru ca un Blueprint să poată utiliza funcțiile definite într-o Blueprint Interface (BPI), este necesar ca interfața respectivă să fie adăugată explicit în secțiunea **Class Settings** [Figură 4.11] a Blueprint-ului, în zona Implemented Interfaces din panoul de detalii. [Figură 4.12] Odată adăugată, funcțiile interfeței devin disponibile pentru implementare și apel în Blueprint-ul respectiv.



Figură 4.11 – Butonul de implementare clase



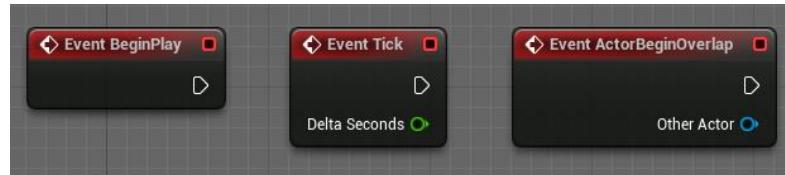
Figură 4.12 – Panoul de detalii al claselor

4.3.3. Noduri fundamentale Blueprint

În cadrul ferestrei Blueprint, nodurile pot fi adăugate prin click dreapta oriunde pe suprafața de editare, moment în care se deschide un meniu contextual. Din acest meniu, utilizatorul poate selecta funcții sau componente dorite, adăugând astfel nodul corespunzător în logica vizuală.

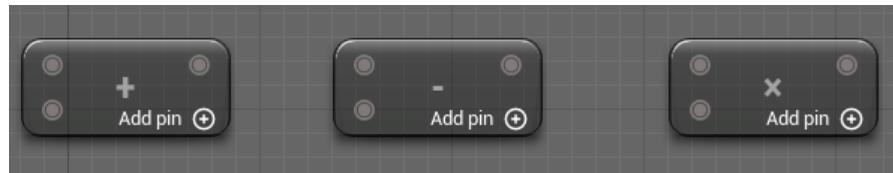
Blueprint-ul pune la dispoziția dezvoltatorilor o multitudine de noduri, organizate logic, care permit construirea rapidă a comportamentelor dorite. Aceste noduri includ:

- **Noduri de event** – evenimente precum **BeginPlay**, **Tick**, **OnComponentHit**, **OnOverlapBegin** care se declanșează automat în anumite momente ale ciclului de viață al unui actor sau ca răspuns la interacțiuni. [Figură 4.13]



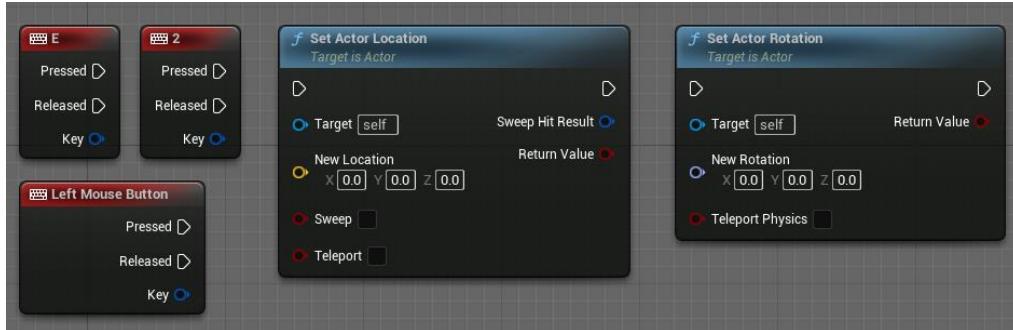
Figură 4.13 – Noduri fundamentale de event

- **Noduri matematice** – adunare, scădere, multiplicare a valorilor, transformări vectoriale și operațiuni trigonometrice. [Figură 4.14]



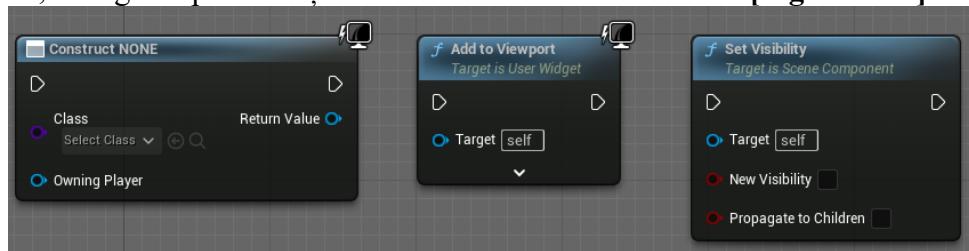
Figură 4.14 – Noduri fundamentale matematice

- **Noduri de mișcare** – **Add Movement Input**, **Set Actor Location**, **Set Actor Rotation** pentru controlul poziției și rotației obiectelor. [Figură 4.15]



Figură 4.15 – Noduri fundamentale de mișcare

- **Noduri de interacțiune UI** – **Create Widget**, **Add to Viewport**, **Set Visibility** pentru crearea, adăugarea pe ecran și setarea de vizibilitate a UI-ului. [Figură 4.16]



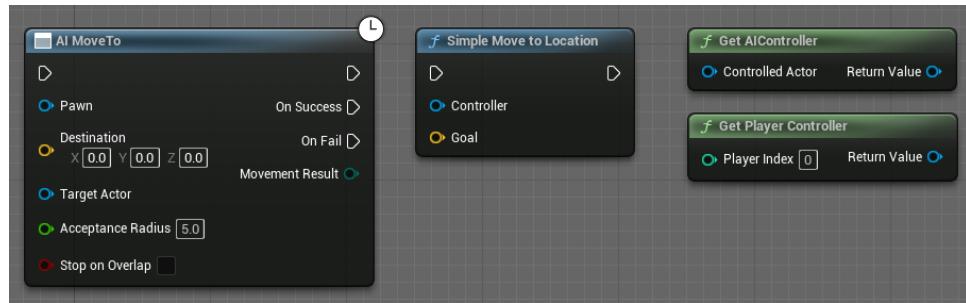
Figură 4.16 – Noduri fundamentale de UI

- **Noduri de timp – Delay, Get World Delta Seconds** pentru așteptări de intervale și contorizare secunde delta. [Figură 4.17]



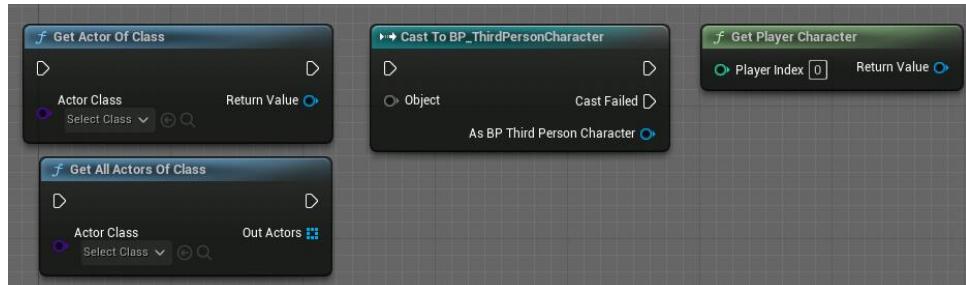
Figură 4.17 – Noduri fundamentale de timp

- **Noduri de control AI – AIMoveTo, Simple Move to Location, Get Controlled Pawn** pentru deplasarea și controlul AI. [Figură 4.18]



Figură 4.18 – Noduri fundamentale pentru controlul AI

- **Noduri de referințe – Get Actor of Class, Cast to Actor, Get Player Character** pentru a permite obținerea de referințe, accesarea metodelor și proprietățile acestora. [Figură 4.19]



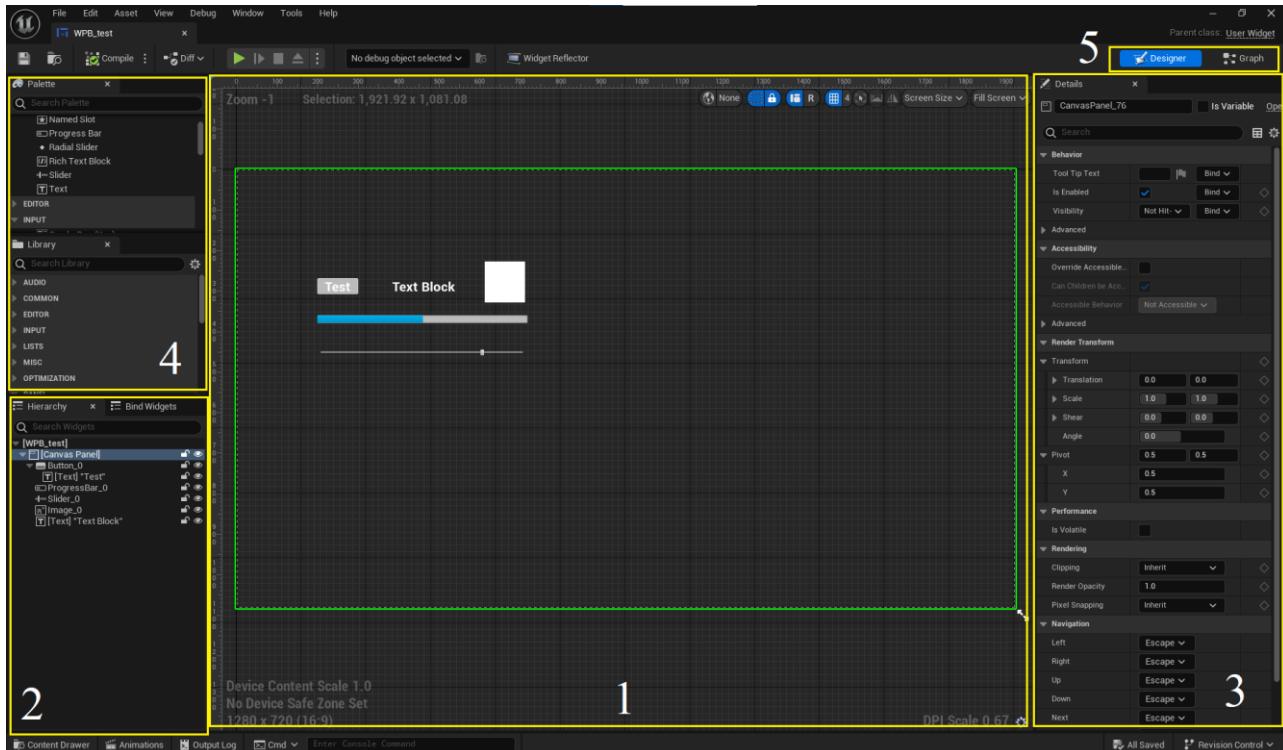
Figură 4.19 – Noduri fundamentale de obținere a referințelor

Și altele.

4.4. Interfață grafică Widget Blueprint

Widget Blueprint este un tip special de Blueprint în Unreal Engine, utilizat pentru crearea de interfețe grafice pentru utilizator. Aceste interfețe pot include meniuri, bări de viață, inventare, ferestre de dialog, butoane sau orice alte elemente vizuale prin care utilizatorul poate interacționa. [18]

4.4.1. Fereastra Widget Blueprint

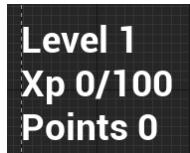


Figură 4.20 – Fereastra Widget Blueprint

1. **Designer View** – zona principală unde sunt plasate și aranjate elementele vizuale ale interfeței (butoane, texte, imagini etc.).
2. **Hierarchy** – afișează structura ierarhică a elementelor UI din widget, facilitând organizarea și relațiile de tip părinte-copil dintre componente.
3. **Details** – oferă acces la proprietățile și setările fiecărui element selectat, permitând personalizarea comportamentului și aspectului acestuia.
4. **Palette** – conține biblioteca de componente UI disponibile (buton, slider, imagine, text, progress bar etc.), din care pot fi selectate și adăugate în widget.
5. **Designer / Graph** – permite comutarea între modul vizual (Designer) și logica de funcționare a widget-ului, definită în **Graph** (folosind noduri Blueprint).

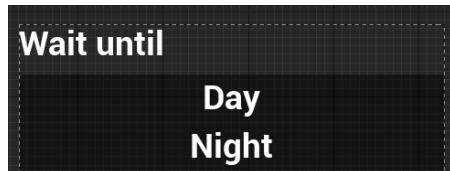
4.4.2. Elementele folosite în cadrul WBP

- **Text Box-uri** – utilizate pentru afişare de informaţie pe ecran (exemplu, statisticile jucătorului). [Figură 4.21]



Figură 4.21 – Componenta text

- **Butoane** – utilizate pentru declanşarea de evenimente legate de interacţiunea cu mediul înconjurător (de exemplu, trecerea de zi-noapte). [Figură 4.22]



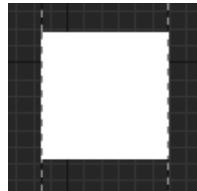
Figură 4.22 – Componenta buton

- **Slidere** – folosite pentru ajustarea valorilor, cum ar fi componentele RGB de culoare a unor texturi de material (de exemplu, armura). [Figură 4.23]



Figură 4.23 – Componenta slider

- **Imagini** – utilizate pentru previzualizarea culorilor selectate în urma setării sliderelor RGB. [Figură 4.22],[Figură 4.24]



Figură 4.24 – Componenta imagine

- **Bări de progres** – utilizate pentru previzualizare (exemplu, procentul de viaţă şi stamina al personajului). [Figură 4.25]



Figură 4.25 – Componenta bară de progres

Şi altele.

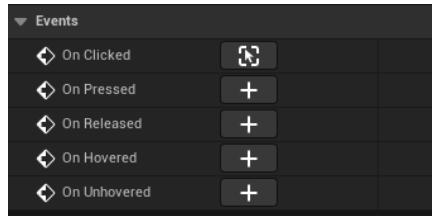
4.4.3. Modul Graph

Pentru gestionarea interacţiunii utilizatorului cu elementele din interfaţa grafică, în cadrul acestui proiect a fost utilizat mecanismul **Event Dispatcher**.

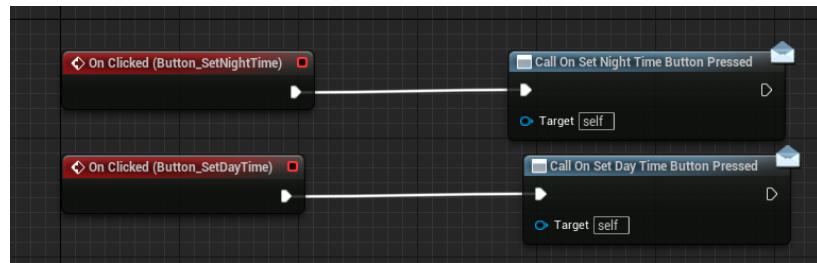
Un Event Dispatcher reprezintă un mecanism care facilitează comunicarea între diferite obiecte din joc, permitând declanșarea unor acțiuni în alte Blueprints fără a crea dependențe directe între ele. Astfel, un element UI, poate emite un semnal (event), iar alte componente ale jocului, abonate la acel dispatcher, vor răspunde în mod corespunzător. [18]

Pentru acest tip de interfață am utilizat:

- **Butoane** cu eveniment de tip **OnClick()** [Figură 4.26], pentru a declanșa acțiuni specifice în momentul interacțiunii utilizatorului, printr-un **Dispatcher**. [Figură 4.27]

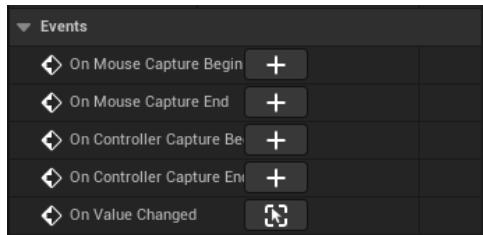


Figură 4.26 – Setarea de eveniment onClick pentru buton

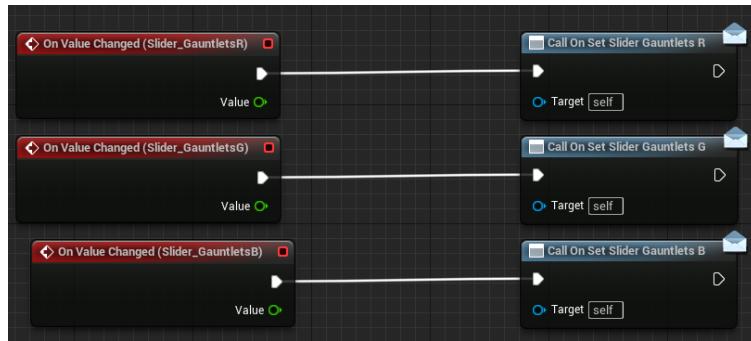


Figură 4.27 – Dispatcher onClick

- **Slidere** cu eveniment de tip **onValueChanged()** [Figură 4.28], pentru a declanșa acțiuni specifice în momentul schimbului valorilor de slider, printr-un **Dispatcher**. [Figură 4.29]



Figură 4.28 – Setarea de eveniment onValueChanged pentru slider



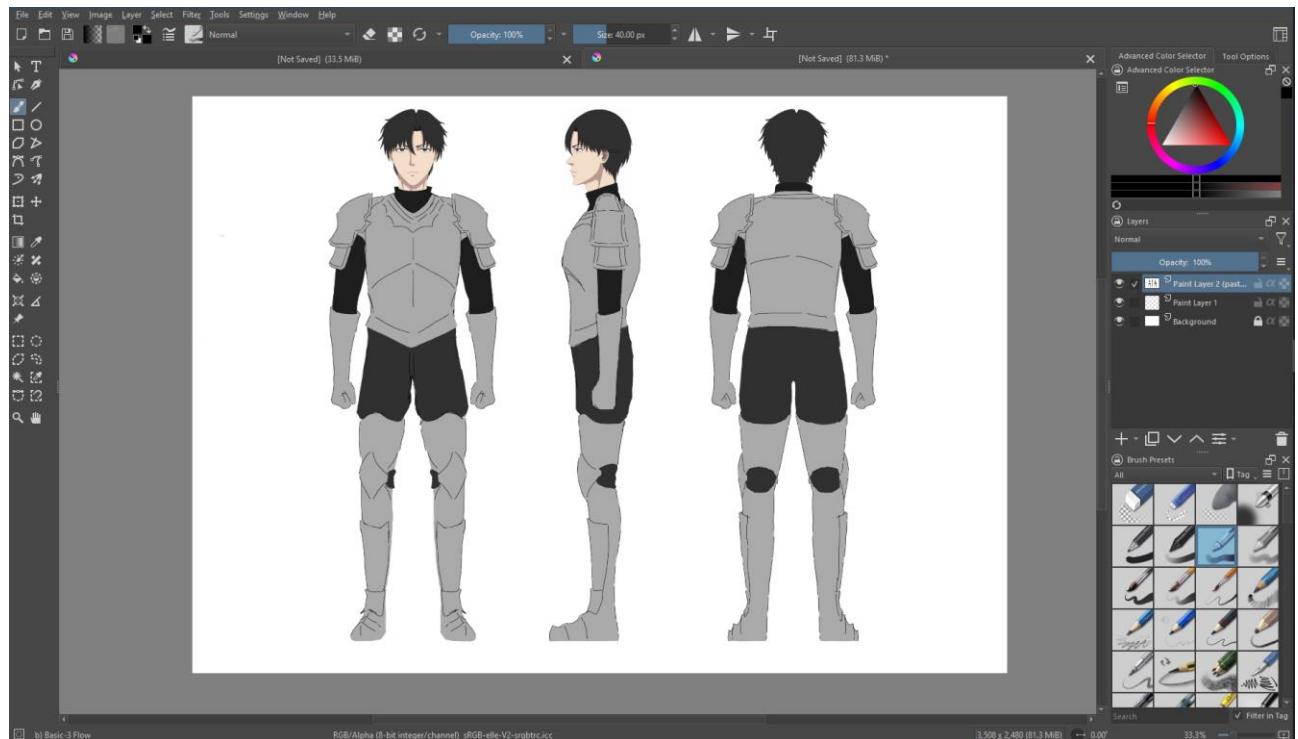
Figură 4.29 – Dispatcher onValueChanged

4.5. Utilizarea aplicației Krita pentru design 2D

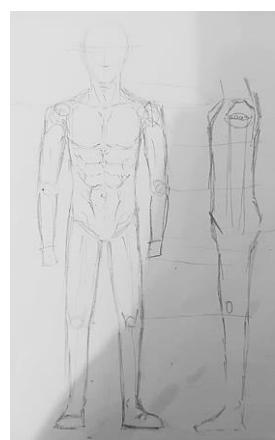
Conceptul de artă (**Concept art**) este procesul de creare al imaginilor vizuale care reprezintă ideile, personajele, decorurile sau atmosfera unui joc, film sau alt proiect creativ, înainte de a fi realizată efectiv în producția 3D sau finală. Practic, concept art-ul servește ca un ghid vizual pentru dezvoltare, oferind o viziune clară și detaliată asupra aspectului și stilului pe care îl va avea produsul final. [19]

Ca și tematică, am ales un univers medieval fantasy care se bazează pe elemente istorice din Evul Mediu, cu peisaje, arhitectură și personaje specifice acelei perioade. Această tematică oferă un cadru narativ potrivit pentru un joc RPG, axat pe explorare, misiuni și interacțiuni într-o lume realistă, dar totodată captivantă.

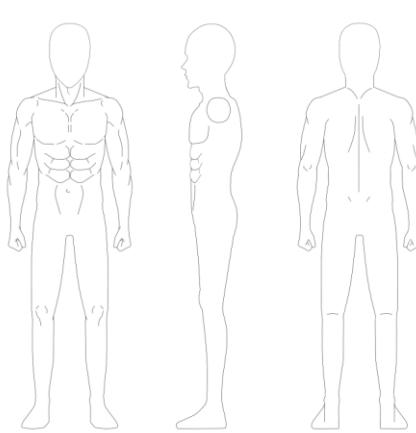
4.5.1. Crearea conceptului vizual al personajului principal



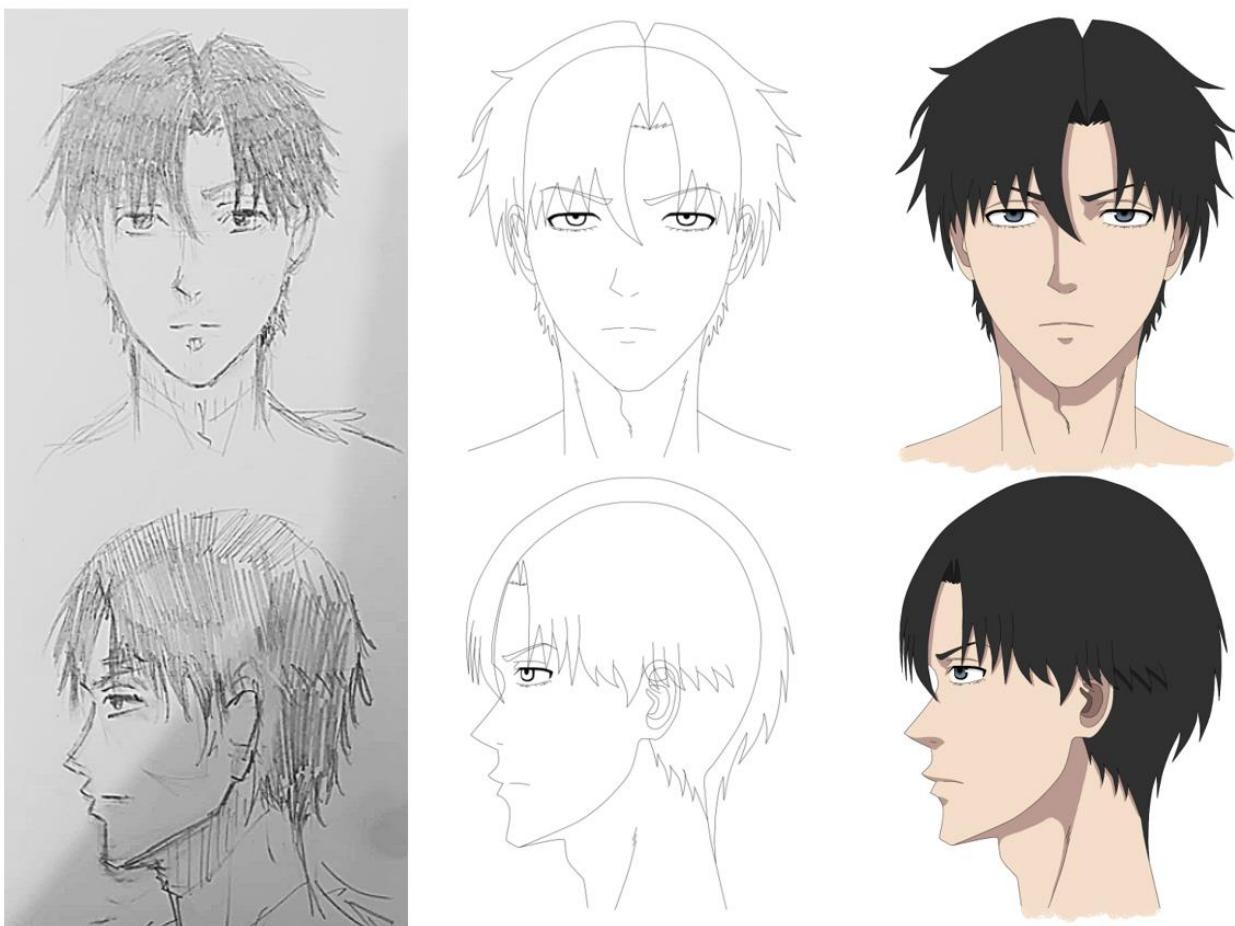
Figură 4.30 - Fereastra de desen & Designul final al personajului principal



Figură 4.31 – Schița personajului principal

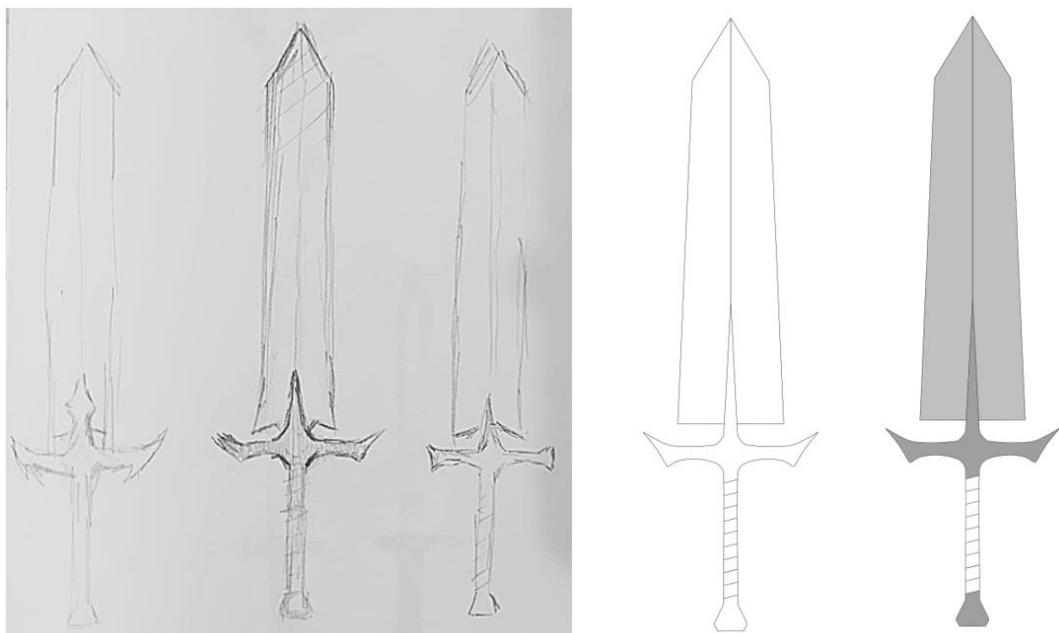


Figură 4.32 – Conturul personajului principal



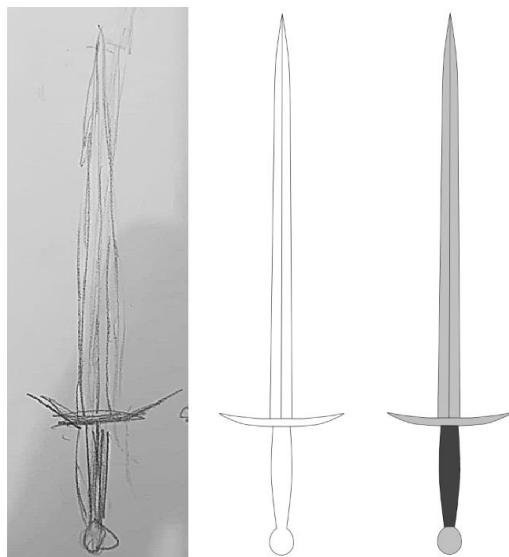
Figură 4.33 – Procesul de concept & design al personajului principal

4.5.3. Crearea conceptului vizual de armă al personajului principal

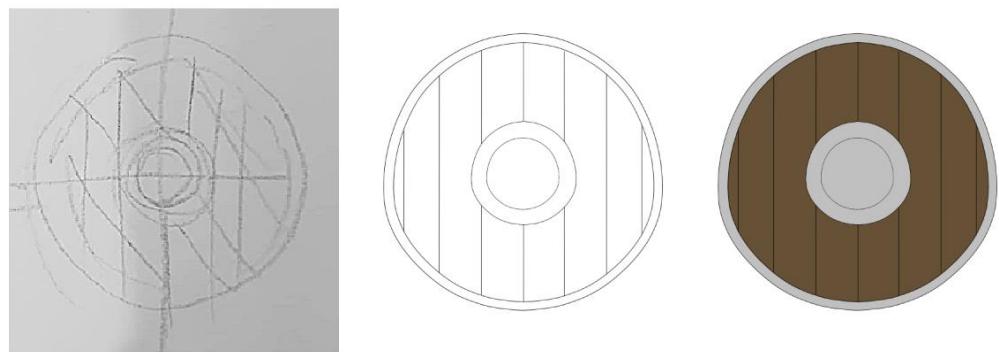


Figură 4.34 –Procesul de concept & design a armei personajului principal

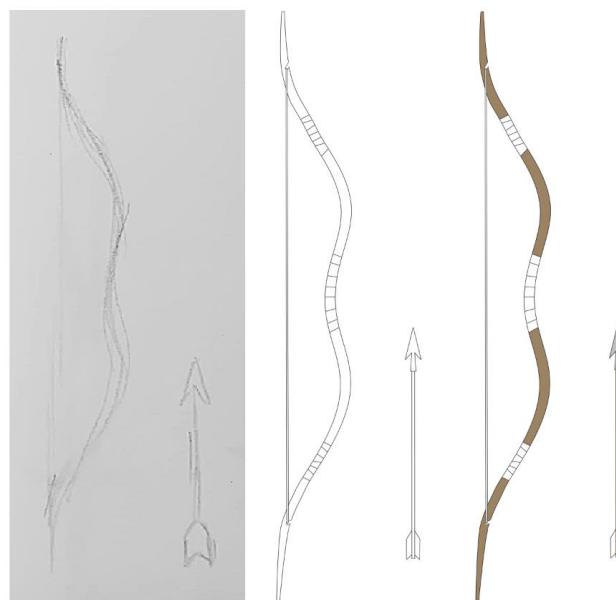
4.5.4. Alte concepte vizuale de design 2D



Figură 4.35 – Procesul de concept & design a armei oponenților AI mele



Figură 4.36 – Procesul de concept & design a scutului oponenților AI mele

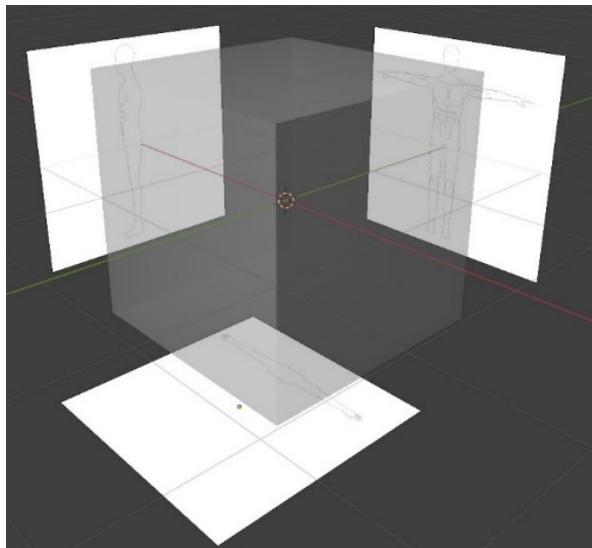


Figură 4.37 – Procesul de concept & design a arcului oponenților AI ranged

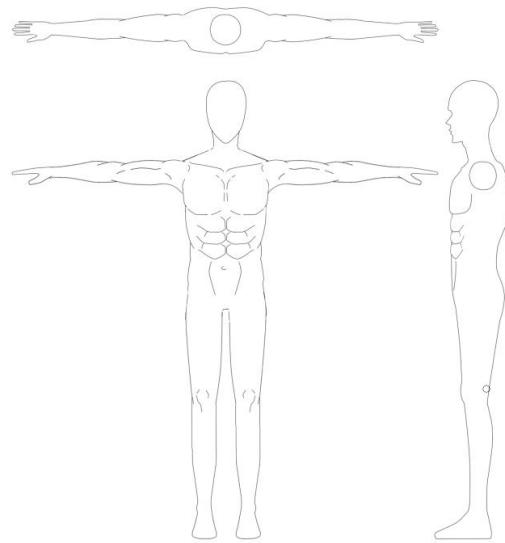
4.6. Utilizarea aplicației Blender pentru modelare 3D

În cadrul proiectului, am utilizat desenele și schitele realizate în etapa de concept art ca bază de lucru pentru procesul de modelare 3D în Blender. Desenele 2D au fost importate și folosite ca referință pentru a construi geometriile tridimensionale ale personajelor, armelor și obiectelor din joc.

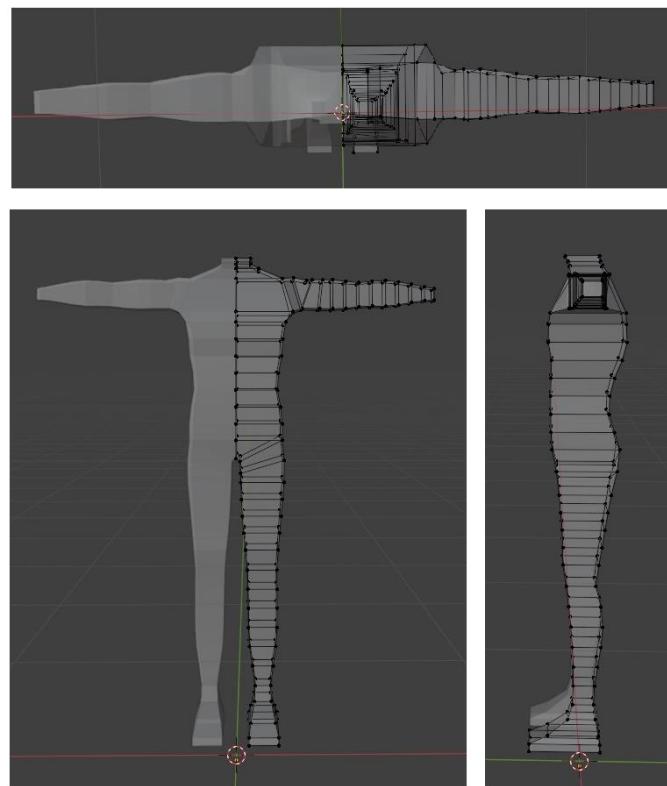
4.6.1. Modelarea personajului principal



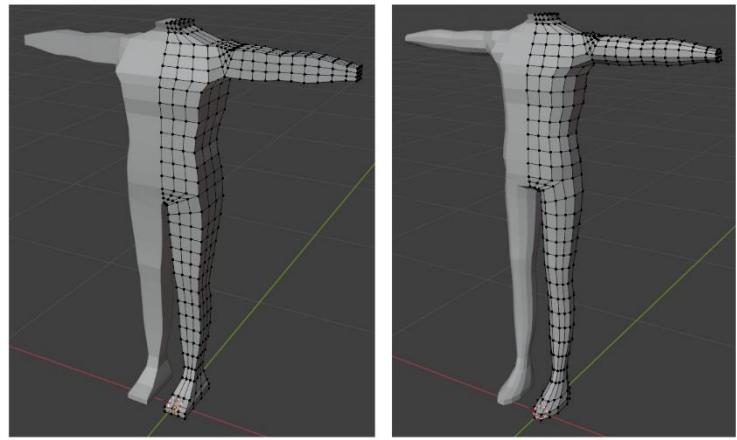
Figură 4.39 – Poziționarea referințelor



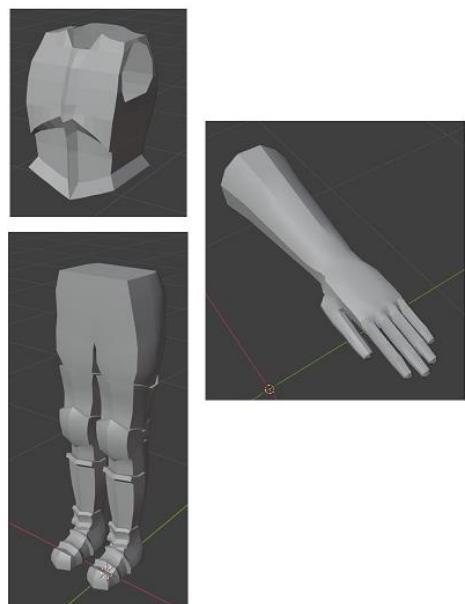
Figură 4.38 – Referința model



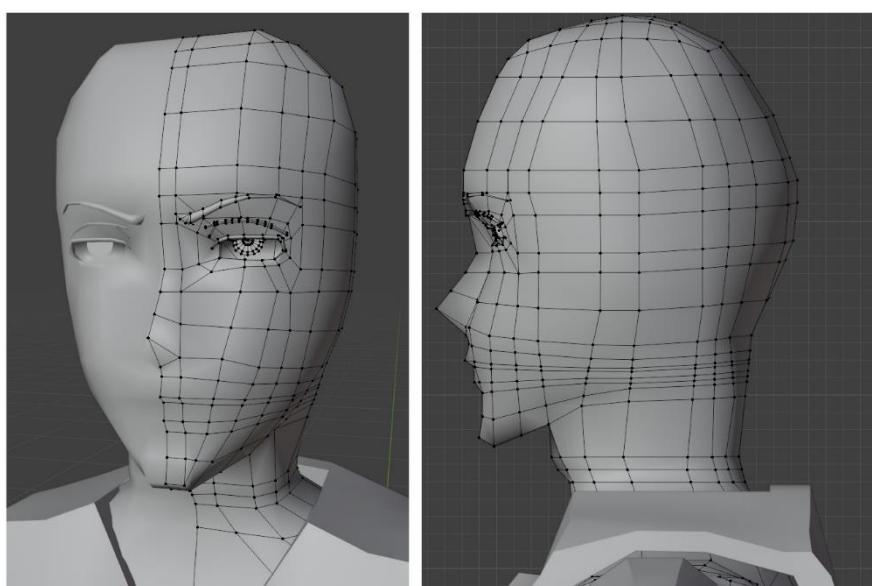
Figură 4.40 – Modelarea 3D a personajului principal



Figură 4.41 – Subdivizarea modelului

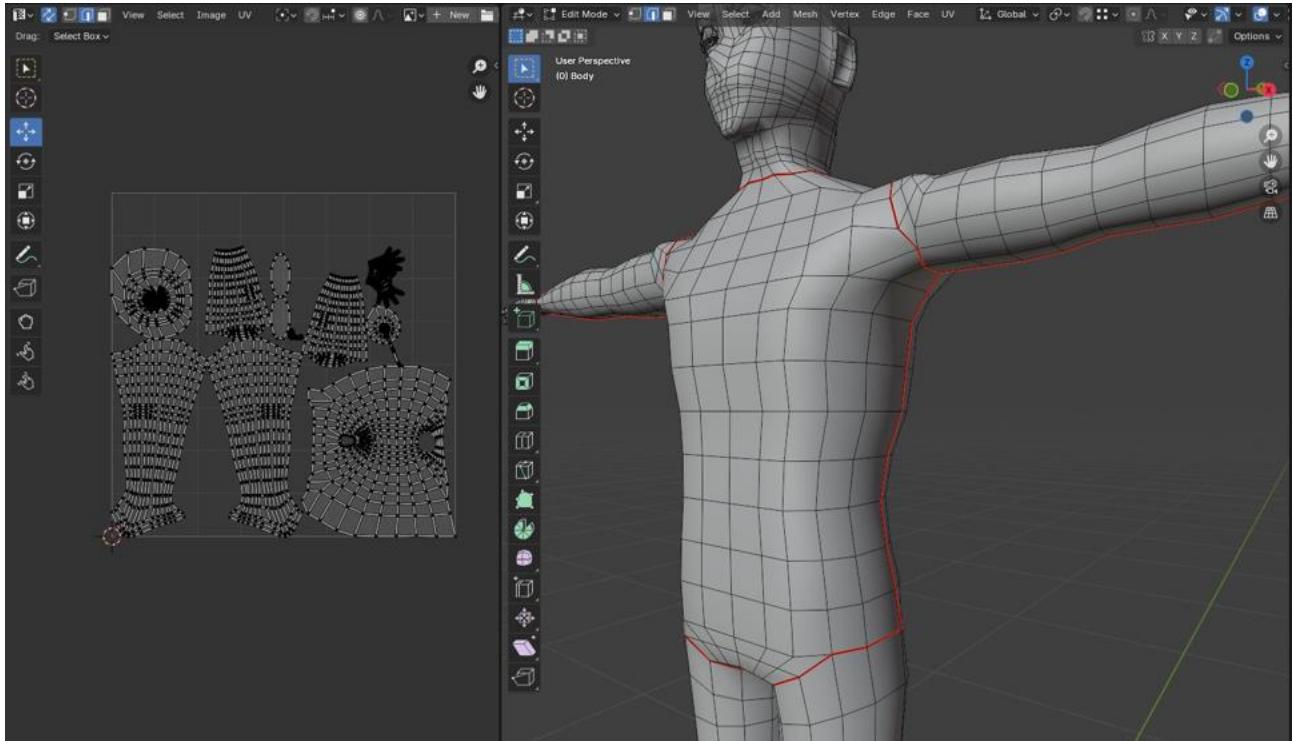


Figură 4.42 – Modelarea pieselor de armură



Figură 4.43 – Modelarea structurii faciale

Procesul de UV Unwrap pentru aplicarea materialelor și texturilor.



Figură 4.44 – Separarea UV-urilor de model



Figură 4.45 - Rezultatul final al modelului 3D

4.6.2. Modele 3D de tip recuzită (Props)

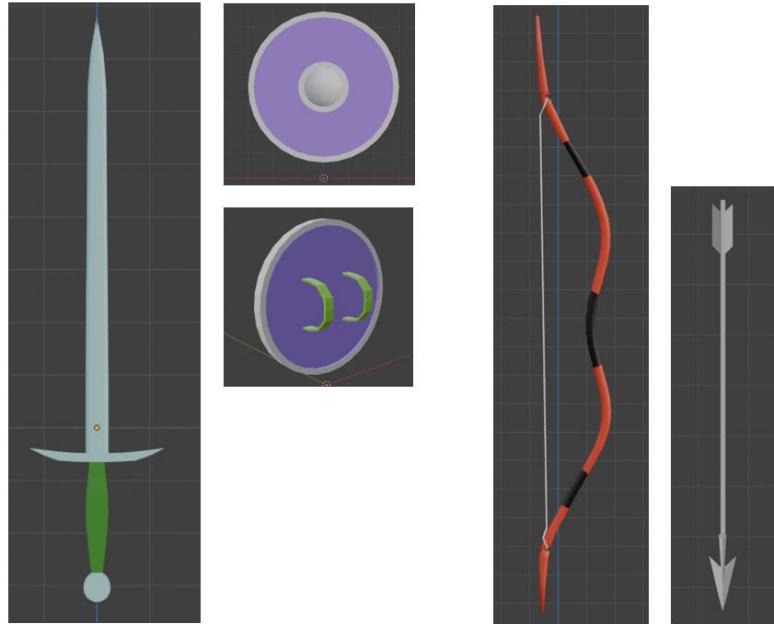
În același mod am proiectat și următoarele modele:

- Modelul de armă al personajului principal. [Figură 4.46]



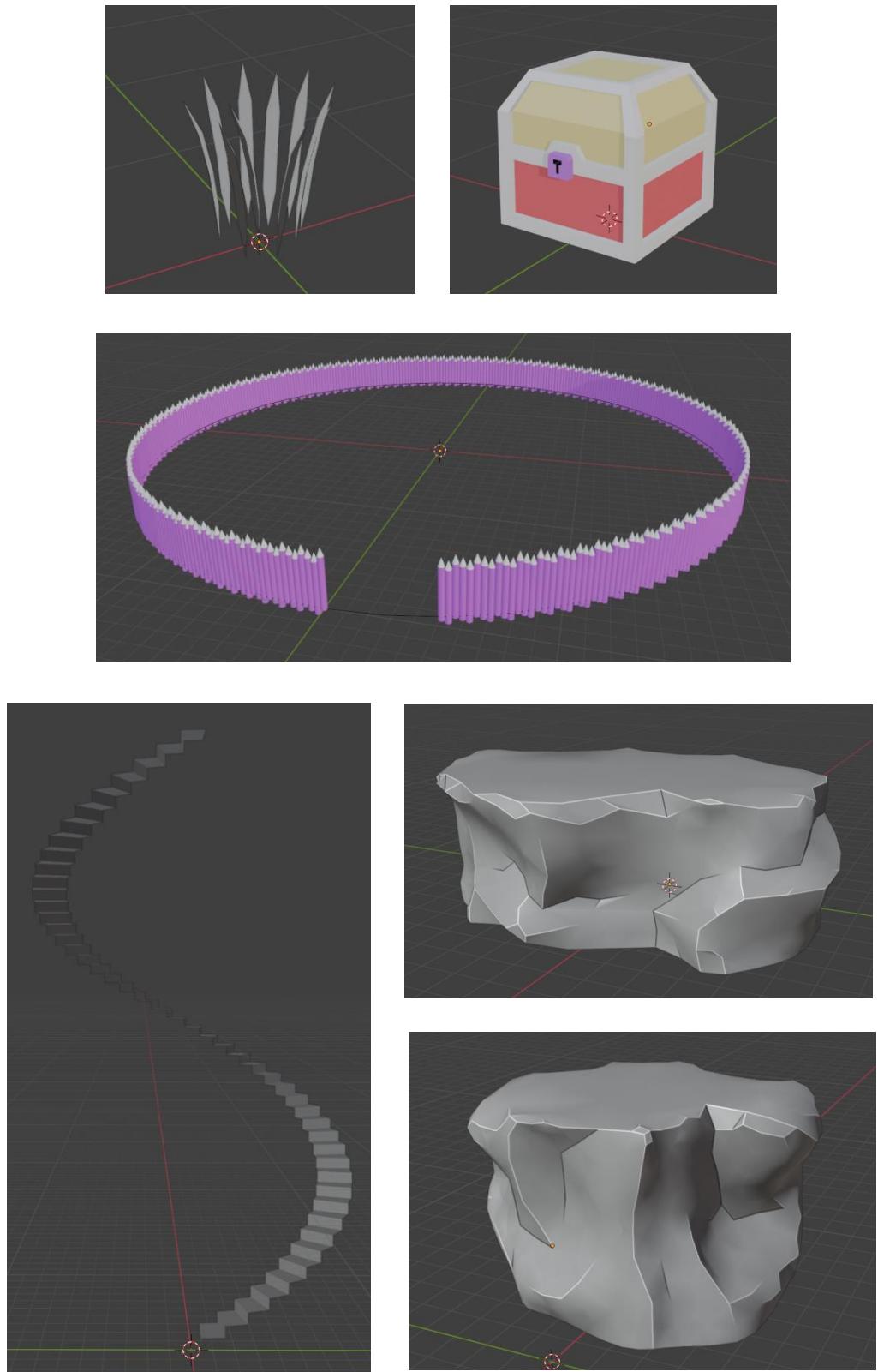
Figură 4.46 – Modelul 3D de armă al personajului principal

- Modelul de armă al oponenților AI Mele și Ranged. [Figură 4.47]



Figură 4.47 – Modele 3D de arme ale oponenților AI, Mele & Ranged

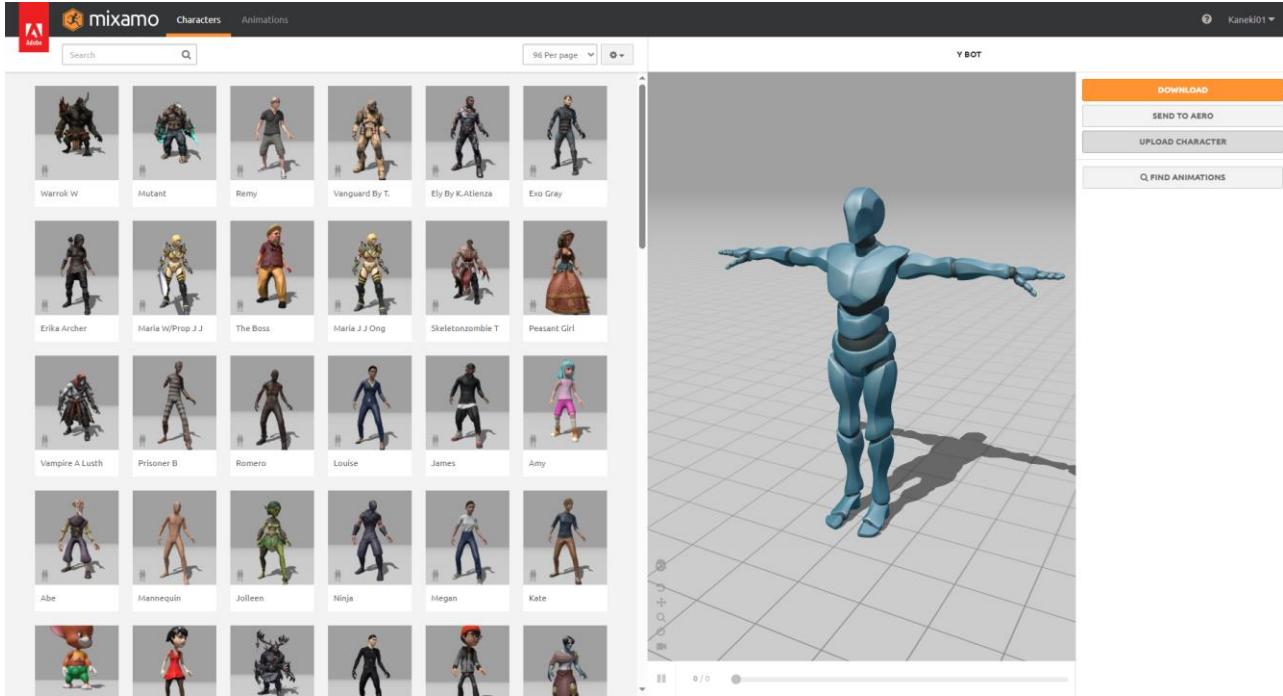
- Alte modele: iarba, cufere, garduri, scări, stânci. **[Figură 4.48]**



Figură 4.48 – Modele de iarba, cufere, garduri, scări, stânci.

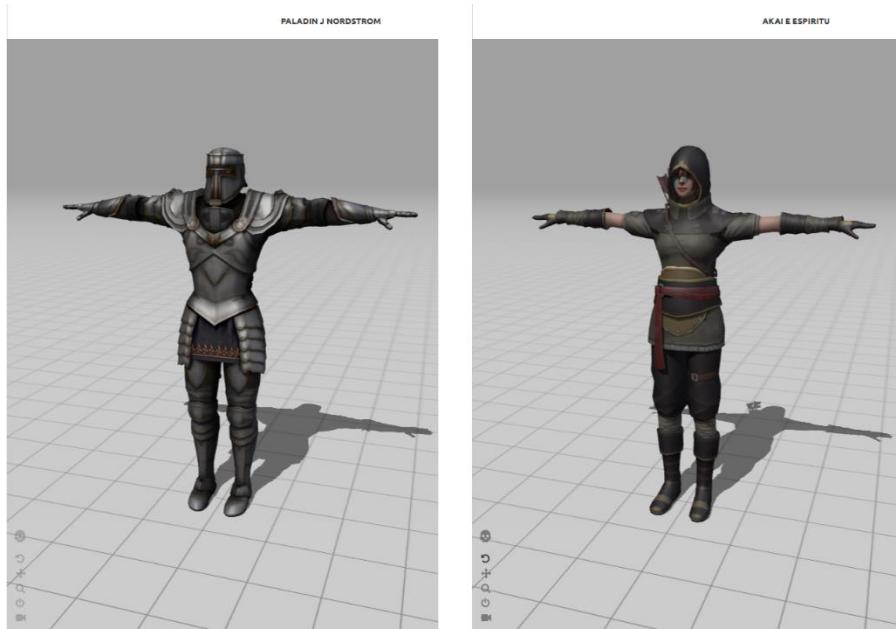
4.7. Utilizarea platformei Mixamo pentru rigging și animații de caractere

Am utilizat platforma Mixamo pentru a adăuga personajul principal și pentru a genera automat un schelet (rig) compatibil. Acest rig a permis integrarea și utilizarea animațiilor disponibile în biblioteca Mixamo, facilitând astfel procesul de animare a personajului în motorul Unreal Engine. [Figură 4.49]



Figură 4.49 – Captură de ecran preluată de pe platforma Mixamo, utilizată cu scop educațional.

Pe lângă personajul principal, au fost folosite și personaje predefinite disponibile pe platformă, împreună cu animații de atac și mișcare de la secțiunea Animation. [Figură 4.50]



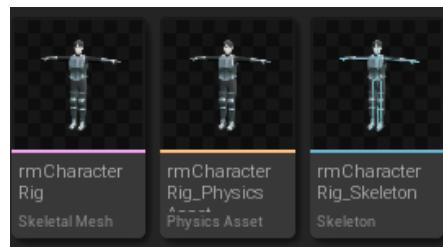
Figură 4.50 – Modele puse la dispoziție de platforma Mixamo

4.8. Sistemul de animații în Unreal Engine

4.8.1. Riguirea personajului principal

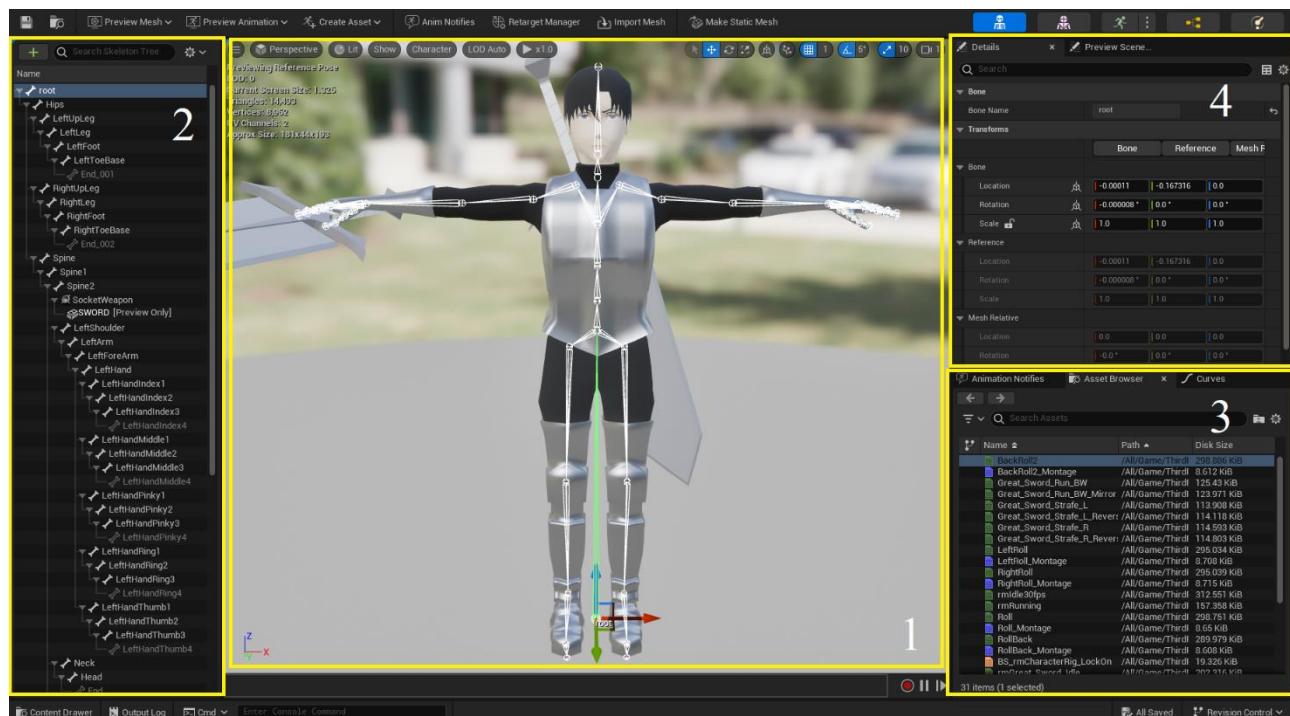
Atunci când un model 3D este importat în Unreal Engine, dacă acesta conține un schelet (rig), motorul generează automat o serie de resurse esențiale [Figură 4.51] pentru utilizarea și animarea acestuia: [20]

- **Skeletal Mesh** – reprezentarea 3D a personajului, legată de structura osoasă (Skeleton), necesară pentru a reda animațiile.
- **Skeleton** – structura ierarhică a oaselor (bones), pe baza căreia este realizată animarea modelului.
- **Physics Asset** – un set de coliziuni și constrângeri fizice (rigid bodies), utilizat pentru simulări fizice precum ragdoll sau interacțiuni dinamice.
- **Animation Sequences** – în cazul în care la import sunt aduse și animații, acestea sunt asociate direct scheletului și salvate ca secvențe de animație individuale.



Figură 4.51 – Resursele scheletului de model

Este important de menționat faptul că, pentru ca un caracter să poată fi animat în Unreal Engine, acesta trebuie să dispună de un schelet valid. [Figură 4.52]



Figură 4.52 – Fereastra de rig și animații a personajului principal

- Preview al scheletului** – permite vizualizarea modelului Skeleton Mesh împreună cu structura oaselor.
- Oasele (Bones)** – afișează ierarhia completă a oaselor din schelet.
- Animațiile personajului (Animation Sequences)** – listă de animații asociate scheletului.
- Panoul de detalii** – oferă posibilitatea de a ajusta poziția, rotația și scara oaselor.

Modelele statice (fără rig), nu pot fi animate prin sistemele dedicate Skeletal Mesh-urilor. Din acest motiv, am utilizat platforma **Mixamo** pentru a genera automat rig-ul necesar pentru personajul principal și pentru alte modele folosite.

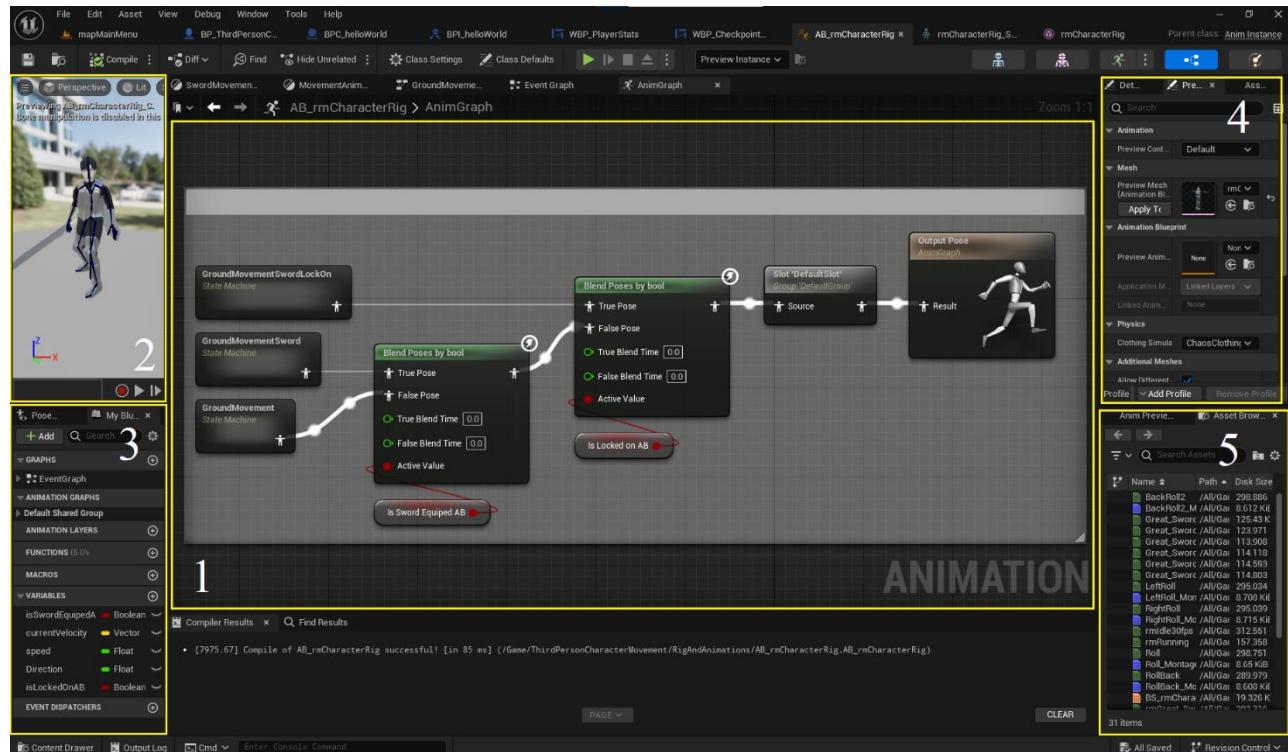
Un aspect esențial al compatibilității animațiilor este faptul că acestea **sunt concepute pentru un anumit tip de schelet**. O animație realizată pentru un schelet diferit poate produce deformări vizuale sau erori în timpul rulării. De aceea trebuie să ne asigurăm că animațiile sunt conform scheletului de bază.

4.8.2. Ce este un Animation Blueprint

Animation Blueprint este un tip specializat de Blueprint în Unreal Engine, dedicat exclusiv controlului și gestionării animațiilor pentru un Skeletal Mesh. Acesta permite definirea logicii de animare a personajului, bazată pe variabile de stare (de exemplu: viteză, direcție, stare de atac), evenimente și condiții din joc. [21]

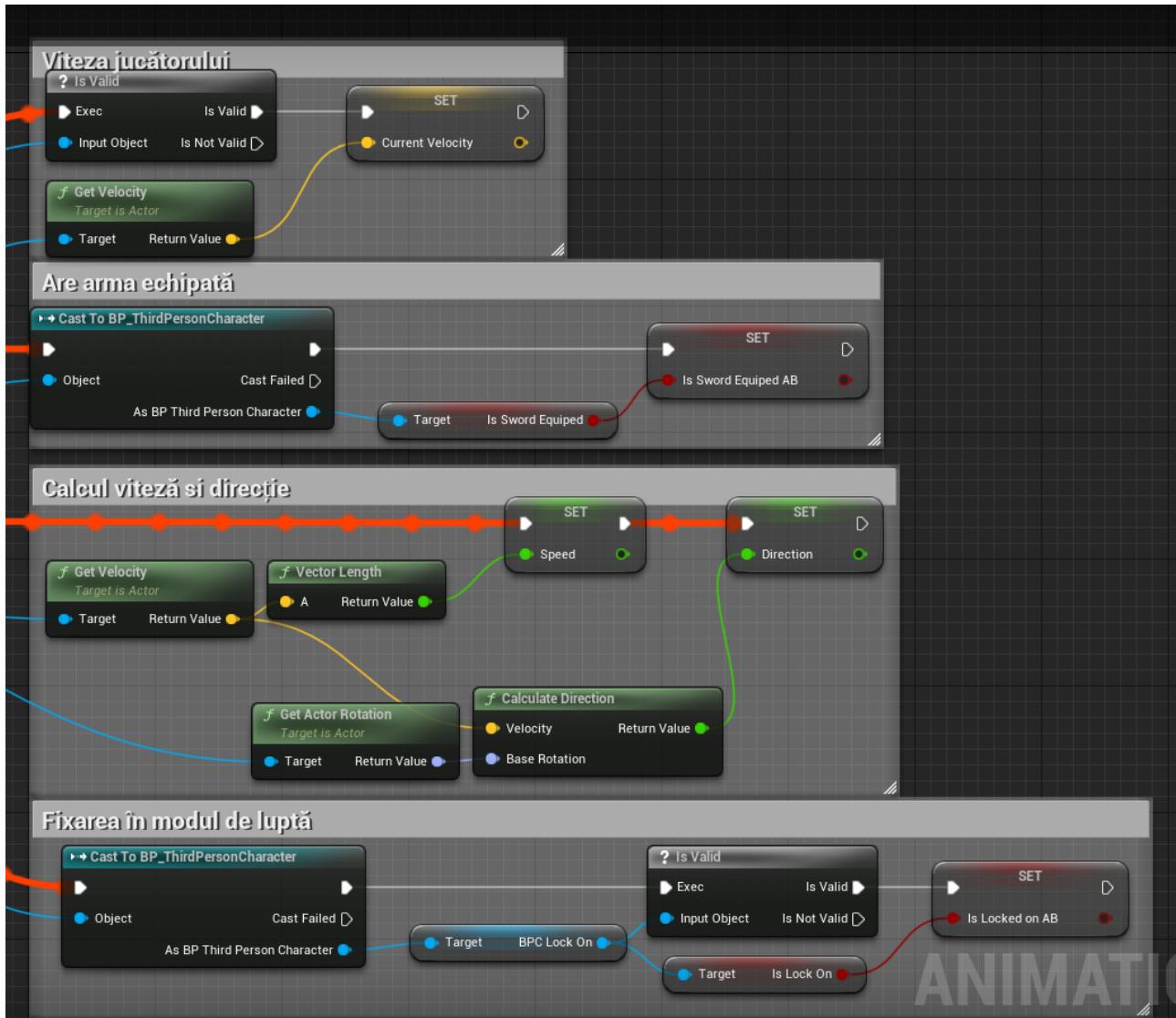
UN ABP este format din două ferestre principale: AnimGraph [Figură 4.53] și EventGraph [Figură 4.53]

- AnimGraph** – este partea grafică a Blueprint-ului unde se definește modul în care animațiile sunt selectate printr-o logică de selecție. Aici se folosesc noduri specifice pentru a crea logica de tranziție între animații (ex: Blend Space, State Machine, Animation Montage). AnimGraph-ul controlează direct rezultatul stărilor și permite doar unei singure animații să ruleze la un moment de timp.



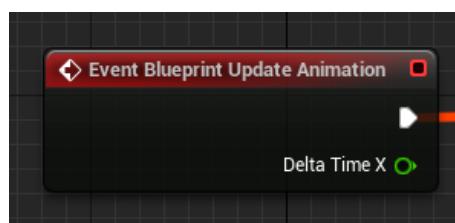
Figură 4.53 – Fereastra AnimGraph

- **EventGraph** – este partea de logică procedurală a Blueprint-ului de animație, unde se definesc evenimente și funcții care actualizează variabilele folosite în AnimGraph. În EventGraph se pot implementa funcții precum obținerea vitezei personajului, verificarea dacă acesta este în aer, procesarea intrărilor de la jucător sau declanșarea anumitor animații contextuale. [Figură 4.54]



Figură 4.54 – Fereastra EventGraph

De obicei, în cadrul EventGraph-ului, funcția **Event Blueprint Update Animation** este folosită pentru a actualiza, la fiecare cadru, valorile parametrilor animației în funcție de starea curentă a personajului. [Figură 4.55]



Figură 4.55 – Nodul de update al variabilelor de animație

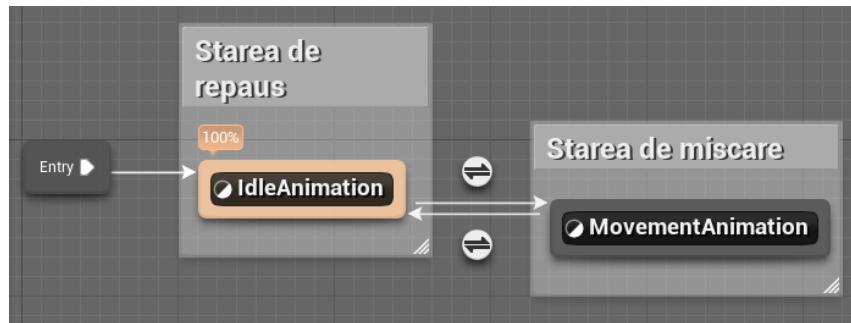
4.8.3. Crearea și utilizarea unui State Machine

Un State Machine controlează tranzițiile dintre diferitele stări de animație ale unui personaj. [Figură 4.56] [22]



Figură 4.56 – Exemplu State Machine

În interiorul unui State Machine [Figură 4.56], se regăsesc mai multe stări, [Figură 4.57] fiecare asociată cu una sau mai multe animații. [Figură 4.58]

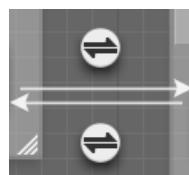


Figură 4.57 – Stările unui State Machine

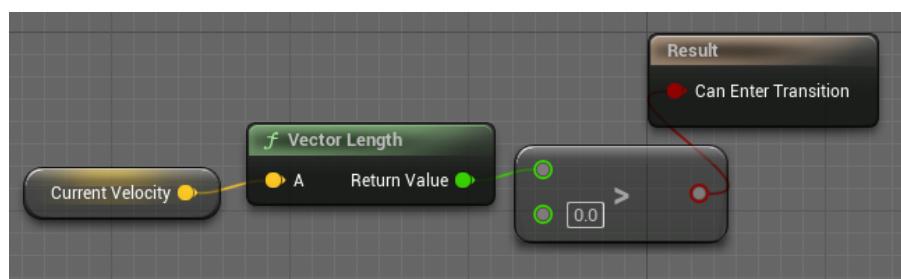


Figură 4.58 – Animatia unei stări

Iar trecerea de la o stare la alta este realizată de condițiile logice, denumite **tranzitii**. [Figură 4.59] Prin intermediul variabilelor din cadrul ferestrei EventGraph [Figură 4.54] și a logicii nodurilor din interiorul acestora. [Figură 4.60]



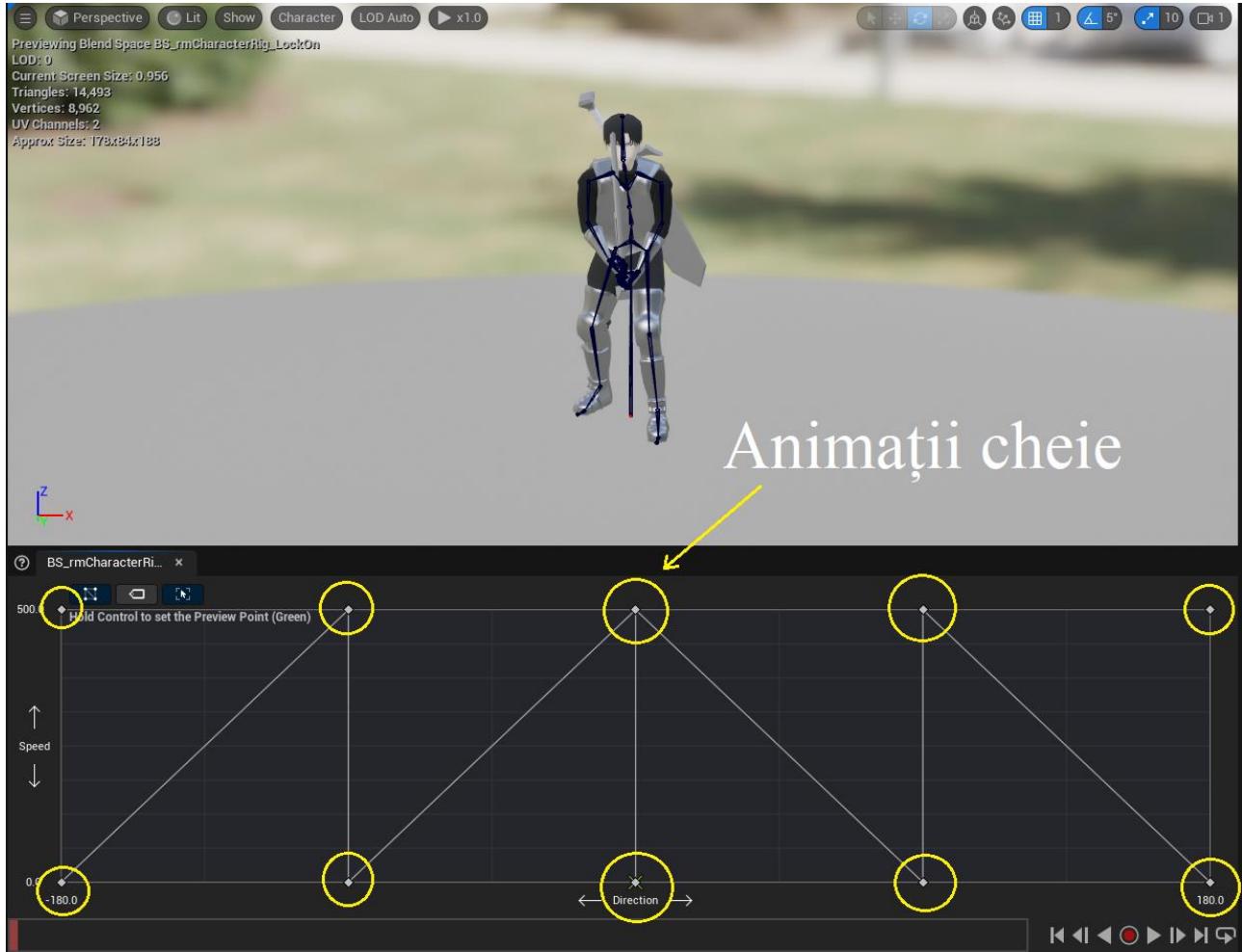
Figură 4.59 – Tranzitiiile stărilor



Figură 4.60 – Exemplu de tranzitie între stări

4.8.4. Animații de tip Blend Space

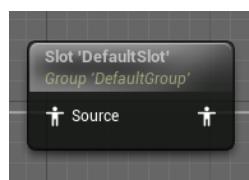
Pe lângă animațiile clasice avem posibilitatea de a crea animații de tip **Blend Space**, care permit combinarea mai multor animații într-o singură, rezultând o tranziție lină între acestea. Un Blend Space funcționează pe baza unor **parametri de control** (precum viteza și direcția de mișcare), în funcție de care se calculează poziția actuală pe un grafic bidimensional și selectarea unei animații. [23]



Figură 4.61 – Animație Blend Space folosită pentru sistemul de Lock-On

4.8.5. Slot-ul de animație Default Slot

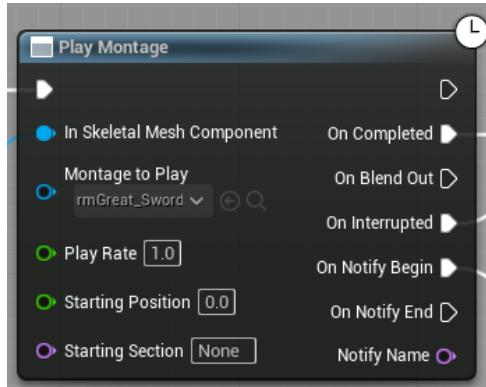
Având în vedere că logica implementată în cadrul State Machine-ului este una simplă, bazată doar pe două stări principale (Run și Idle), am utilizat nodul **Default Slot** pentru a reda **animații intermedii**. [Figură 4.62] [24]



Figură 4.62 – Nodul Default Slot

4.8.6. Montaje de animații (Animation Montage)

Animation Montage-urile sunt active derivate din animațiile normale, dar care oferă un control mai avansat asupra redării acestora. Ele permit definirea unor secvențe de animații care pot fi redate independent de State Machine, fiind ideale pentru acțiuni punctuale sau contextuale, precum atacuri, interacțiuni, sărituri speciale sau reacții ale personajului. [Figură 4.63] [25]



Figură 4.63 – Exemplu de nod Animation Montage

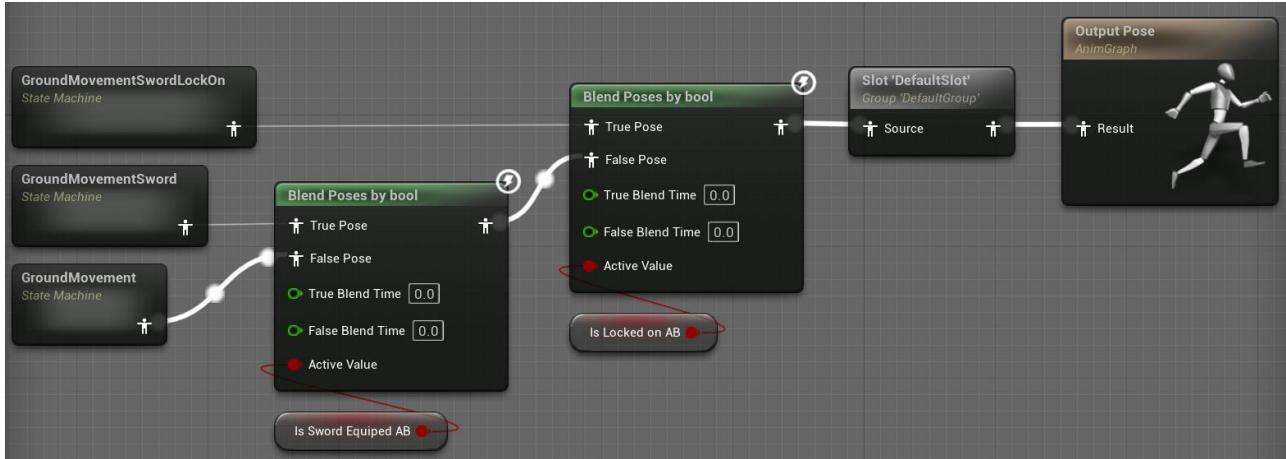
4.8.7. Condiții de selectare State Machine

Permit comutarea între animații sau state machin-uri pe baza unei variabile de tip boolean. [Figură 4.64]



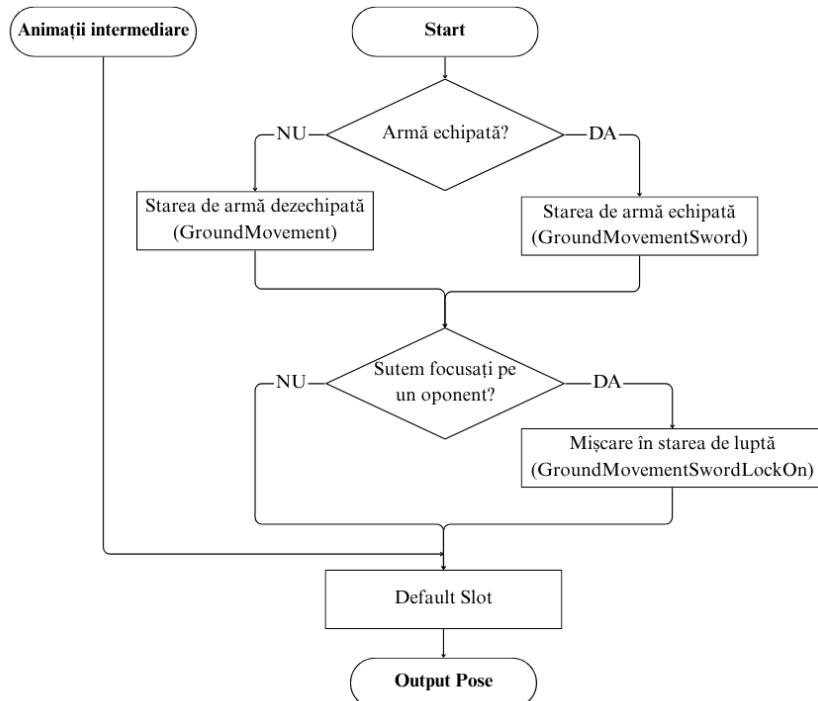
Figură 4.64 – Nodul de Blend Pose

4.8.8. Blueprint-ul de Animație al personajului principal

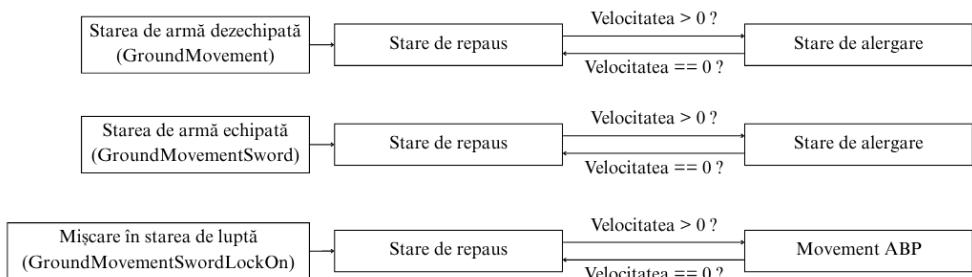


Figură 4.65 – AB a personajului principal

Organograma generală a Blueprint-ului de Animație a personajului principal



Figură 4.66 – Organograma AB a personajului principal

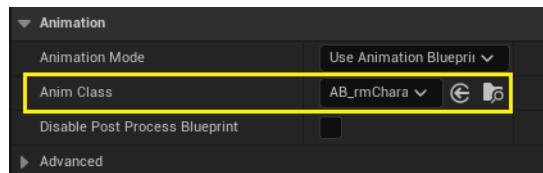


Figură 4.67 – Stările mașină ale personajului principal

Organograma descrie selecția de state machine activ la un moment de timp în funcție de acțiunile pe care jucătorul le realizează. [Figură 4.66] Personajul jucătorului este alcătuit din trei state machines care îndeplinesc animațiile diferite în funcție de contextul acestora. [Figură 4.67]

- **GroundMovement** – activarea animațiilor stării de armă dezechipată.
- **GroundMovementSword** – activarea animațiilor stării de armă echipată.
- **GroundMovementSwordLockOn** – activarea animațiilor stării de luptă.

După realizarea Blueprint-ului de Animatie îl vom atribuii clasei jucătorului la secțiunea „Class Default”, secțiunea **Animation**, parametrul **Anim Class**. [Figură 4.68]



Figură 4.68 – Clasa de atribuire AB a personajului principal

4.9. Utilizarea Sistemului AI din Unreal Engine

Sistemul AI este conceput pentru a determina controlul componentei de inteligență artificială. Acesta oferă un set de unelte și componente care facilitează dezvoltarea unor comportamente complexe, realiste și dinamice. [26]

Componentele principale ale AI în Unreal

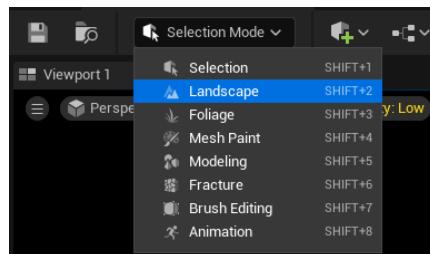
- **AI Controller** – Un tip special de controller care controlează comportamentul unui personaj non-jucabil (NPC). Preia decizii și controlează mișcarea, atacurile sau alte acțiuni.
- **Behavior Trees** – Structuri vizuale care definesc logica de decizie a AI-ului, permitând crearea unor stări de acțiune în funcție de context.
- **Blackboard** – este o „memorie” pentru AI, care stochează date temporare utilizate în Behavior Trees, cum ar fi locația țintei, starea curentă, sau variabile utile decizionale.
- **Perception System** – Sistemul care permite AI-ului să „vadă” mediul înconjurător, detectând jucători, obiecte sau alte entități relevante.
- **Navigation System** – Sistemul de navigație care permite AI-ului să se deplaseze eficient în lumea jocului, folosind navmesh-uri și căutări de cale.

4.10. Landscape

Landscape (peisajul), este componenta principală care alcătuiește jocului din Unreal Engine. Acesta permite crearea de medii vaste și detaliate, câmpii, dealuri sau văi, oferind un spațiu realist și natural gameplay-ului. [27]

Caracteristici principale ale Landscape

- **Modelare și sculptare** – posibilitatea de modelare a terenului folosind unelte de sculptură, ridicând sau coborând zone pentru a crea un relief de joc dorit. [Figură 4.69]



Figură 4.69 – Selectarea modului Landscape

- **Texturare și materiale** – posibilitatea de adăugare material și texturi variate pentru a da aspectul potrivit terenului (iarbă, copaci, zăpadă etc.).
- **Foliage și Vegetație** – Integrează automat vegetație precum copaci, iarba sau arbuști, pentru a popula terenul. [Figură 4.70]



Figură 4.70 – Foliage iarba

Iarba este o componentă static mesh, însă o putem converti în foliage pentru a reduce resursele consumate și a optimiza performanța.

5. Implementarea aplicației

5.1. Sistemul de control al jucătorului

Sistemul de control al jucătorului reprezintă una dintre componentele esențiale ale unui joc video. Acesta gestionează acțiunile transmise de către un jucător sub formă de input-uri prin intermediul unui personaj virtual și primirea de răspuns din partea unor evenimente, permitând astfel interacțiunea directă cu mediul digital al jocului.

Sistemul va fi implementat sub forma unui **Blueprint Class Actor**, fiind alcătuit din mai multe sub-sisteme care lucrează împreună pentru a oferi o experiență de joc interactivă. Printre acestea se numără:

1. Sub-sistemul de control

- Mișcare și orientare
- Interacțiune

2. Sub-sistemul de luptă

- Atacuri Modulare
- Lock-on
- Trasări de linii
- Daune jucător – AI

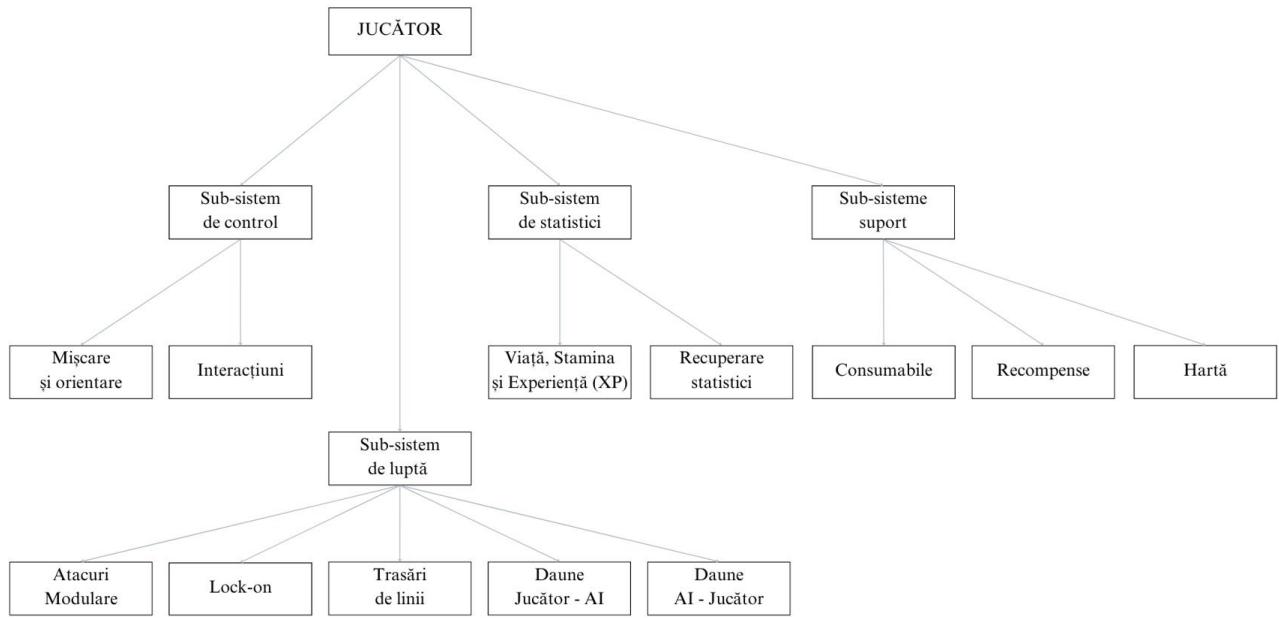
3. Sub-sistemul de atribute

- Viață, Stamina și Experiență (XP)
- Recuperare atribute

4. Sub-sisteme suport

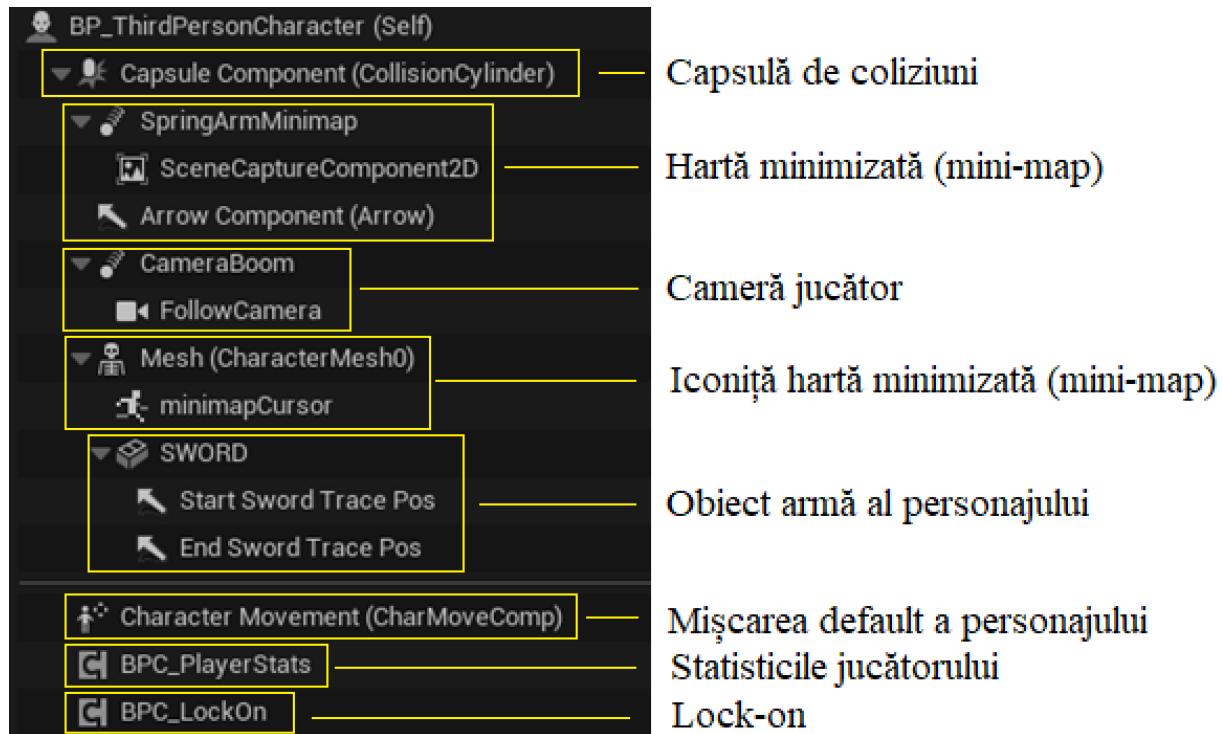
- Consumabile
- Recompense

ORGANIGRAMA GENERALĂ A SISTEMULUI DE CONTROL AL JUCĂTORULUI



Figură 5.1 – Organigramă generală a sistemului de control al jucătorului

Componentele care alcătuiesc Actorul jucător



Figură 5.2 – Componența de alcătuire ale actorului jucător

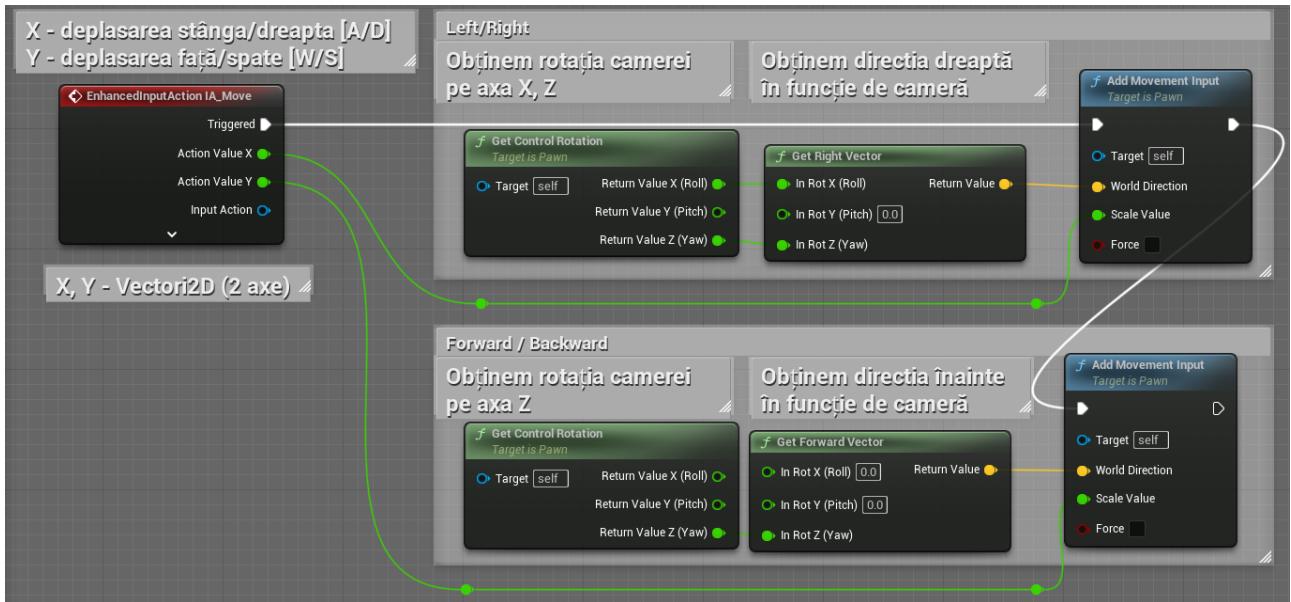
Le vom acoperi pe toate pe parcurs, în cele ce urmează.

5.1.1. Sub-sistemul de control

Sub-sistemul de control permite jucătorului să trimită comenzi către personajul virtual, denumit **Pawn**, gestionând mișcarea, rotația și interacțiunile acestuia cu mediul din joc, pe baza unor input-uri de la tastatură și mouse.

Mișcarea și Orientare

Controlul direcției de deplasare



Figură 5.3 – Controlul direcției de deplasare (Blueprint)

Unreal Engine ne pune la dispoziție această funcționalitate prin template-ul predefinit. Totuși, este important să înțelegem modul în care funcționează de fapt acesta.

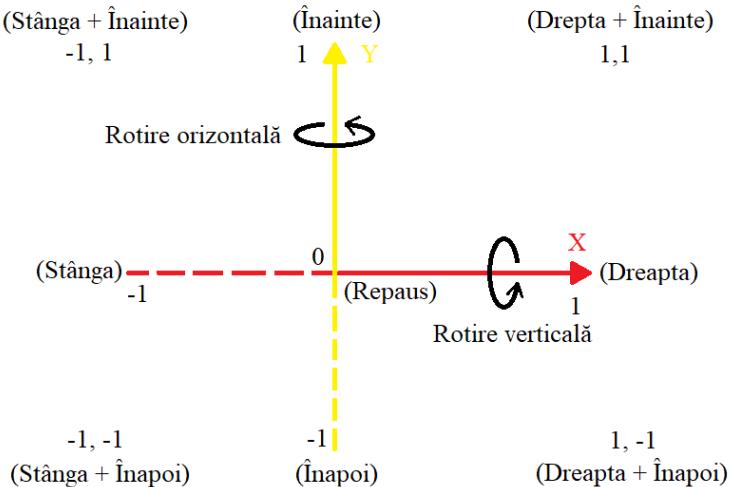
X și Y sunt 2 variabile de tip **Vector2D** care pot lua valori cuprinse între [-1:1] în funcție de tipul de input (tastatură sau controller analogic).

În cazul unui controller analogic, sensibilitatea joystick-ului oferă valori fine asupra mișcării, în timp ce tastatura lipsită de această funcționalitate, va transmite doar valori de -1 și 1.

- **Variabila X** controlează direcția de deplasare a personajului **stânga-drepta**, dar permite deplasarea și pe **diagonală**, în combinație cu Y.
- **Variabila Y** controlează direcția de deplasare a personajului în **înainte-înapoi**, în funcție de orientarea camerei. **Cu alte cuvinte, rotația personajului este influențată de direcția de orientare a camerei controlată de către jucător.**

Aceste valori sunt conectate la funcția **AddMovementInput()** pentru a actualiza direcția de deplasare a personajului.

Axele de deplasare ale jucătorului

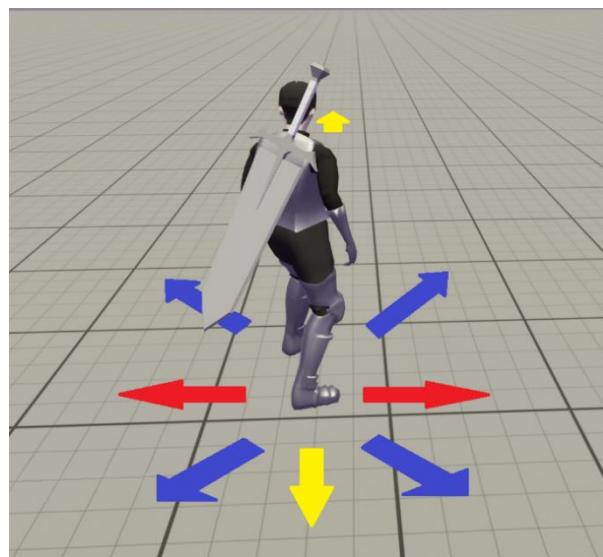


Figură 5.4 – Axele de deplasare ale jucătorului XY

Tabelul de Interpretare al valorilor pentru determinarea direcției de deplasare

X	Y	Input-uri taste	Direcție
0	0	-	Repaus
0	1	W	Înainte
0	-1	S	Înapoi
1	0	D	Dreapta
1	1	D + W	Dreapta + Înainte (Diagonală)
1	-1	D + S	Dreapta + Înapoi (Diagonală)
-1	0	A	Stânga
-1	1	A + W	Stânga + Înainte (Diagonală)
-1	-1	A + S	Stânga + Înapoi (Diagonală)

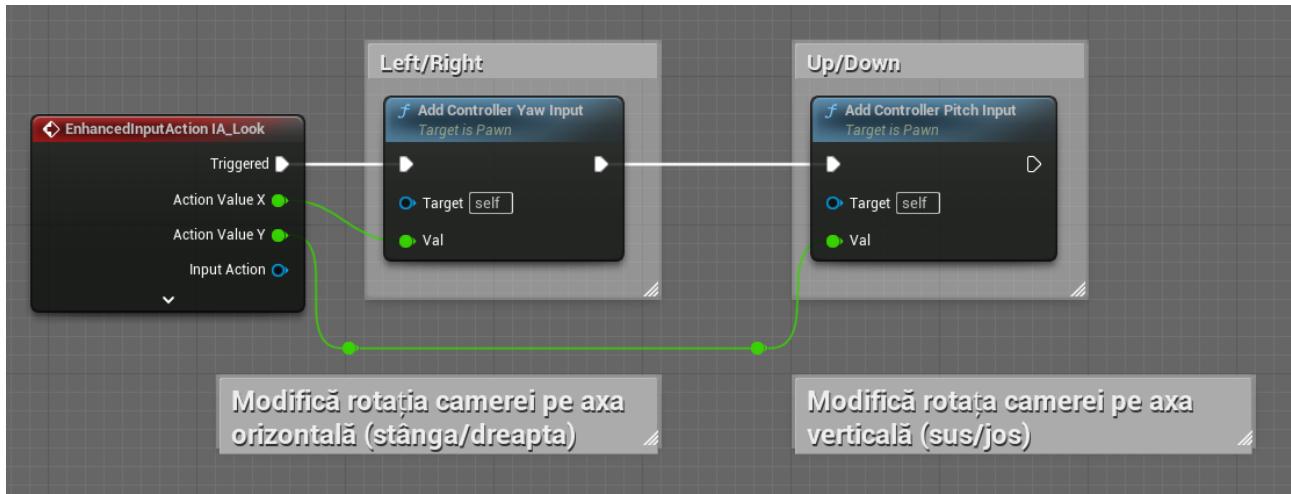
Tabel 5.1 – Tabelul de interpretare al valorilor pentru determinarea direcției de deplasare



Figură 5.5 – Reprezentare vizuală a axeelor de deplasare ale jucătorului XY

Controlul orientării camerei

Controlul orientării este gestionată de **componenta camerei**, aceasta preia input-urile de la mouse și ajustează unghiul de vedere al jucătorului.



Figură 5.6 – Controlul orientării camerei (Blueprint)

Actualizarea constantă este îndeplinită la fiecare mișcare a mouse-ului pe care o execută jucătorul prin următoarele funcții:

- **AddControllerYawInput()** – controlează rotația pe axa orizontală, permitând jucătorului să orienteze camera spre stânga sau dreapta.
- **AddControllerPitchInput()** – controlează rotația pe axa verticală, permitând jucătorului să orienteze camera în sus sau jos.

Ambele funcții primesc ca argument variabilă de tip float cu valori între [-1:1], care reprezintă gradul de intensitate și rotație în funcție de mișcarea mouse-ului sau al unui controller analogic. [Tabel 5.2]

Tabelul funcțiilor de orientare ale camerei

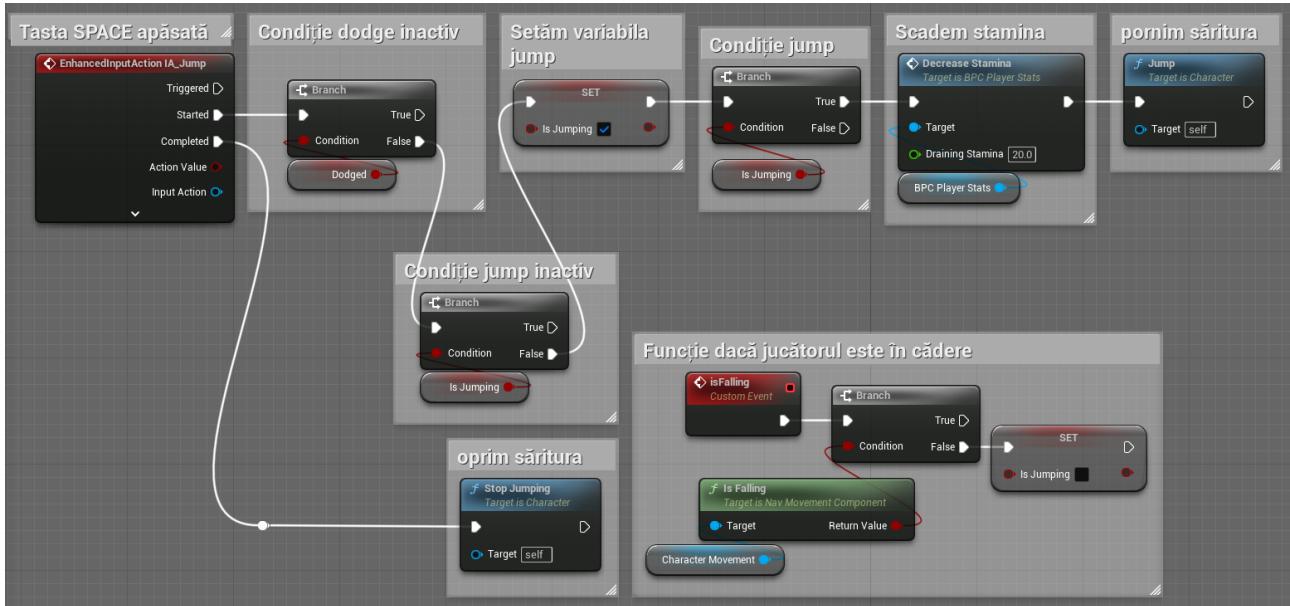
Funcție	X	Direcție
AddControllerYawInput()	X>0	Dreapta
	X<0	Stânga

Funcție	Y	Direcție
AddControllerPitchInput()	Y>0	Sus
	Y<0	Jos

Tabel 5.2 – Tabelul funcțiilor de orientare ale camerei

Funcționalitatea de Săritură (Jump)

Săritura (Jump) este o funcționalitate esențială, oferind varietate și dinamism experientei de joc. La apăsarea tastei „Space” aceasta permite sărituri peste obstacol, care în mod normal, nu ar fi fost posibile prin simplul sistem de control al direcției. [Figură 5.7]



Figură 5.7 – Funcționalitatea de săritură Blueprint

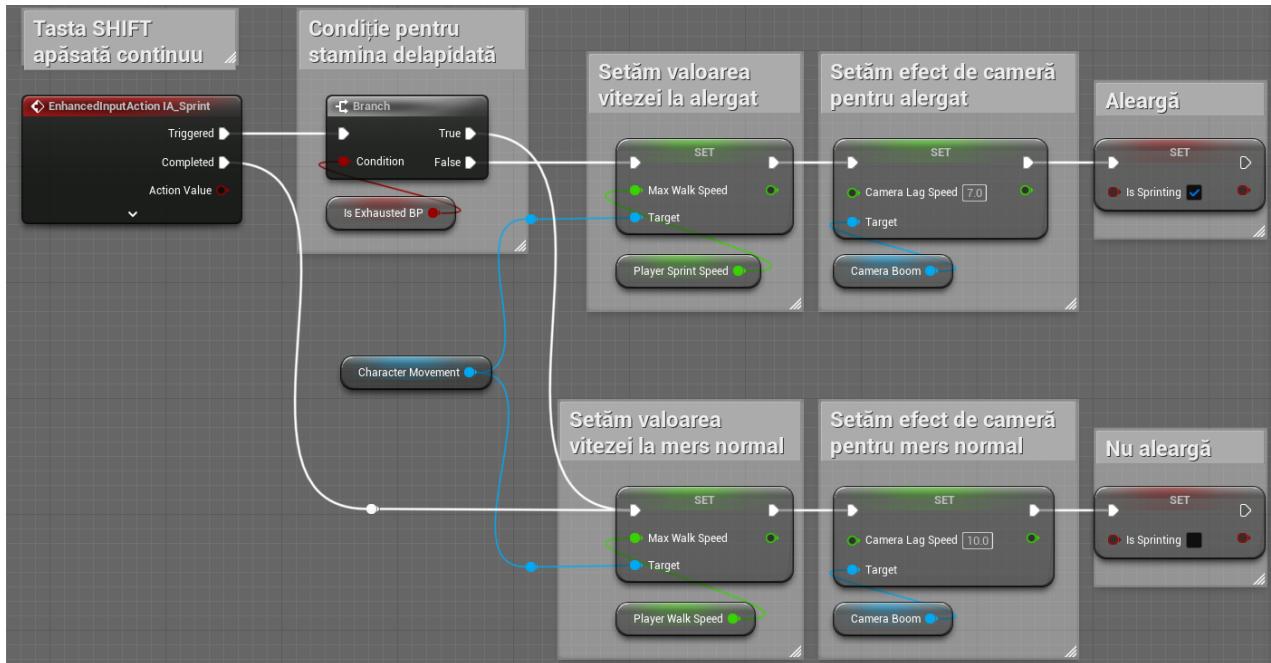
```

La acționarea tastei „SPACE” (Started)
  IF (dodge == False), THEN
    | IF (isJumping == False), THEN
    |   | isJumping = True
    |   | IF (isJumping == True), THEN
    |   |   | Execută decreaseStamina(20.0)
    |   |   | Execută Jump()
    |   | END IF
    | END IF
  END IF
  Când acțiunea este finalizată (Completed)
  Execută StopJumping()
  
```

- **Dodge** – este o **variabilă booleană** care tine evidență dacă jucătorul se află într-o stare de evitare la un moment dat. Această variabilă este utilizat pentru a preveni suprapunere altor acțiuni, în cazul nostru, săritura. Nu ar fi realist ca personajul să poată evita în timp ce acesta se află în starea de săritură. Bineînteles, există diferite tipuri de jocuri care permit și această funcționalitate suplimentară, totuși, pentru a păstra coerentă și logica mecanicilor din joc am ales să o excludem.
- **isJumping** – este o **variabilă booleană** care verifică dacă în momentul actual personajul sare sau nu.
- **Jump()** – funcția este folosită pentru aplicarea unei forțe de deplasare verticală pe axa Z (în sus) a **coordonatelor spațiului 3D**. Prin setarea unei variabile bool **bPressedJump** dacă nu se află deja în aer.
- **StopJumping()** – funcția oprește săritura în momentul în care tasta a fost eliberată, resetând variabila **bPressedJump**, pentru a înceta din aplicare a forței verticale și revenirea la poziția solului.
- **isFalling()** – funcția verifică în permanență la fiecare moment de timp dacă personajul este în cădere sau nu pentru a nu permite sărituri multiple.

Funcționalitatea de Alergare (Sprint)

Această funcționalitate are rol de a permite deplasare jucătorului cu o viteză mai mare pentru o anumită perioadă de timp prin apăsarea continuă a tastei „Shift”. [Figură 5.8]



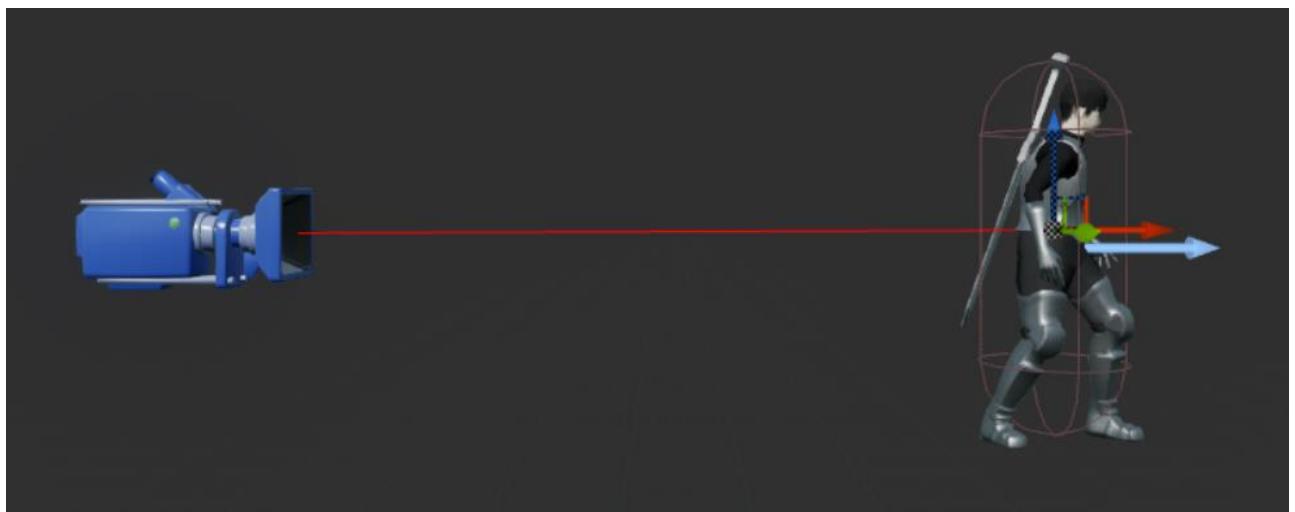
Figură 5.8 – Funcționalitatea de alergare (Blueprint)

```

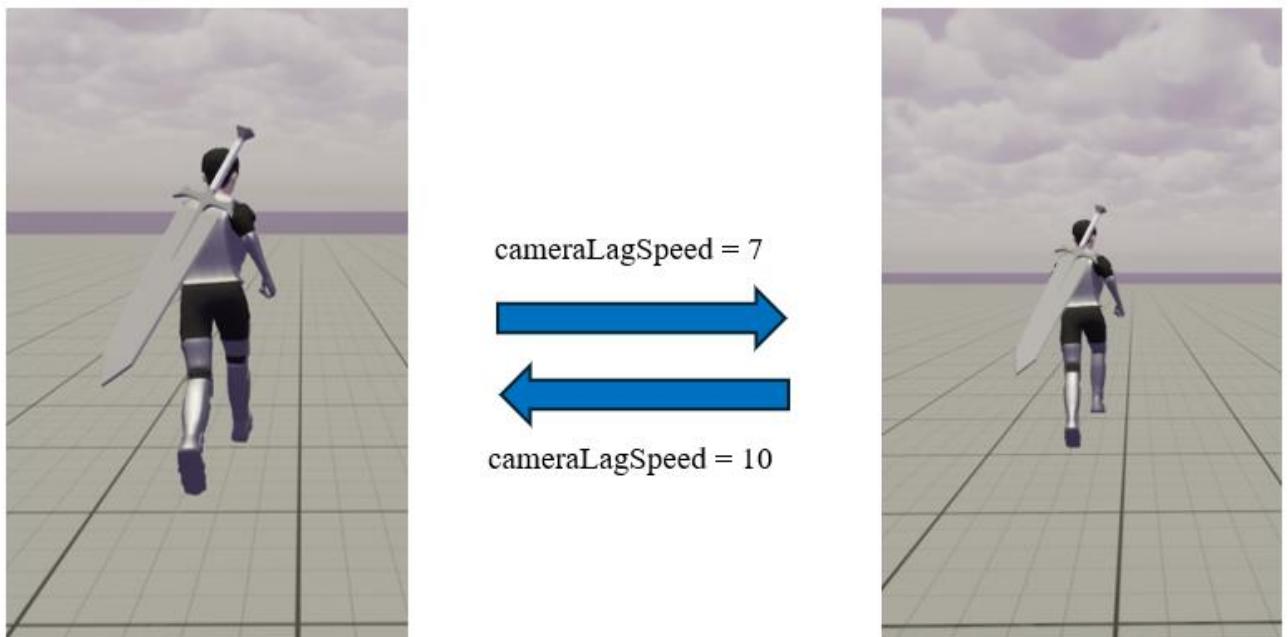
La apăsarea continua a tastei „SHIFT” (Triggered)
IF (isExhausted == False), THEN
    maxWalkSpeed = playerSprintSpeed //800
    cameraBoom.cameraLagSpeed = 7
    isSprinting = True
ELSE (La eliberarea tastei „SHIFT” (Completed) OR isExhausted == True)
    maxWalkSpeed = playerWalkSpeed //500
    cameraBoom.cameraLagSpeed = 10 //valoare default
    isSprinting = False
END IF

```

- **isExhausted** – este o variabilă booleană, folosită pentru a indica epuizarea de stamna a personajului.
- **maxWalkSpeed** – este o variabilă de tip float care setează viteza curentă de deplasare a personajului. Alternează între **playerSprintSpeed** și **playerWalkSpeed** în funcție de context, dacă aleargă sau nu.
- **cameraLagSpeed** – este o variabilă de tip float a componentei **brațului de Camera (Camera Boom)** [Figură 5.9], care setează viteza camerei de urmărire a personajului. Cu cât valoarea este mai mică, cu atât camera se va mișca mai lent, creând un efect de întarziere vizual (lag), în timp ce o valoare mai mare face ca mișcarea camerei să fie cât mai precisă cu deplasarea personajului. [Figură 5.10], [Figură 5.11]
- **isSprinting** – variabilă booleană utilizată pentru a determina dacă personajul se află în starea de alergare (sprint), TRUE sau mers (walk), FALSE.



Figură 5.9 – Brațul de cameră



Figură 5.10 – Reprezentarea vizuală a vitezei de cameră cu lag

Functionalitatea de Evitare (Dodge)



Figură 5.11 – Funcționalitatea de evitare (Blueprint)

```

Prin apăsarea tastei „CTRL” (Pressed)
  IF ((canDodge == True) AND (NOT isFalling == True)), THEN
    IF (keyPressedBackwards == True), THEN
      Execută decreaseStamina(20.0)
      canDodge = False
      dodged = True
      Execută animația de Dodge înapoi
      delay(1.0)
      dodged = False
      canDodge = True
    ELSE
      Execută decreaseStamina(20.0)
      canDodge = False
      dodged = True
      Execută animația de Dodge înainte
      delay(1.0)
      dodged = False
      canDodge = True
    END IF
  END IF

```

- **canDodge** – este o variabilă booleană care validează posibilitatea de a evita un atac primit de la un oponent.
- **isFalling** – variabila verifică dacă personajul se află în cadere.
- **keyPressedBackwards** – este o variabilă de tip booleană verifică dacă tasta „S” a fost apăsată sau nu.
- **dodged** – este o variabilă de tip bool care semnalează faptul că personajul a evitat sau nu.
- **delay()** – funcție care primește ca și argument o variabilă de tip float și impune un timp de așteptare până să se execute următoarea instrucțiune.

5.1.2. Interacțiuni

Interacțiunile reprezintă acțiuni pe care jucătorul le poate realiza asupra obiectelor sau personajelor pentru a influența mediul sau progresul din joc. Tipul de interacțiune, diferă de context și abordarea aleasă.

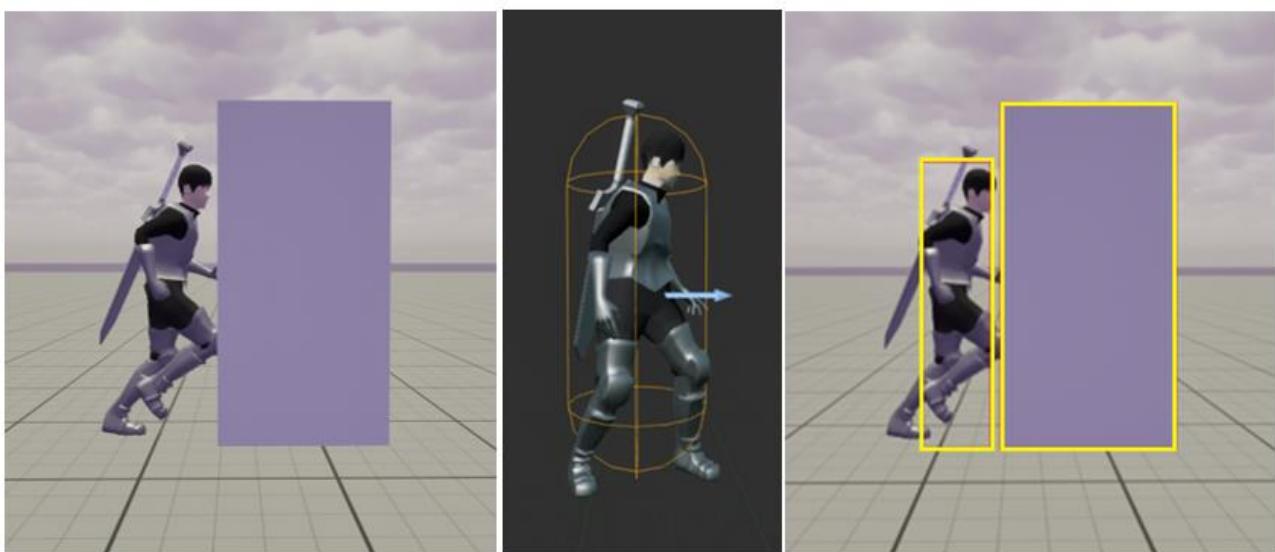
Au fost implementate următoarele tipuri de interacțiuni:

- Interacțiuni cu obiecte prin coliziuni.
- Interacțiuni cu personaje prin coliziuni.

Interacțiuni cu obiecte prin coliziuni

Acest tip de interacțiune este utilizat pentru a detecta contactul fizic dintre personaj și obiectele mediului virtual, având ca scop declanșarea unor comportamente specifice sau limitarea de acțiuni. Cum ar fi:

- **Oprirea deplasării personajului** dacă acesta are în față un obiect, datorită sistemului de coliziuni pe care îl are deja implementat Unreal Engine prin **componenta capsulă**, Capsule Component. [Figură 5.12]

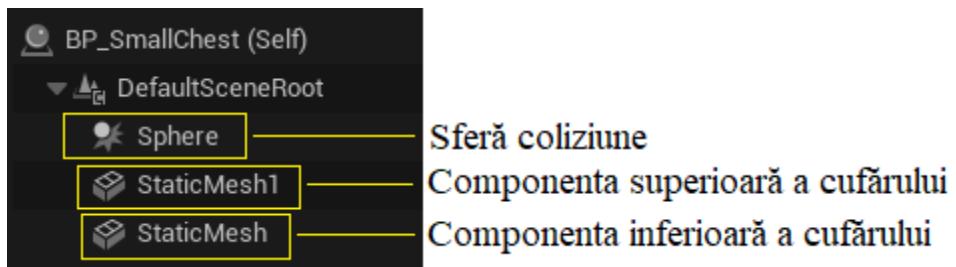


Figură 5.12 – Reprezentarea vizuală a coliziunilor cu obiecte

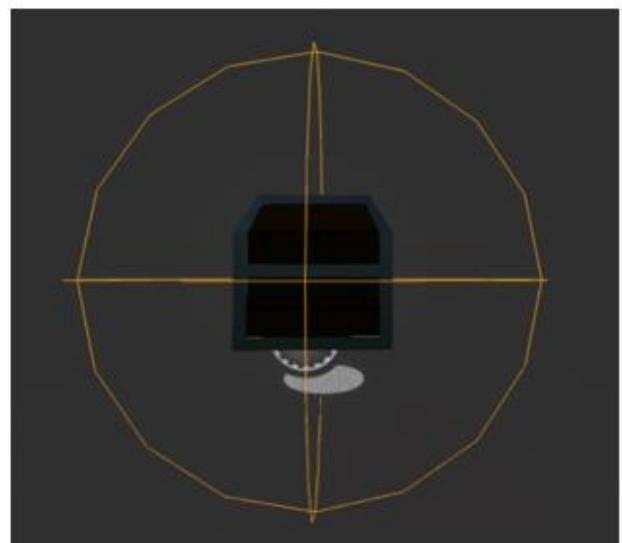
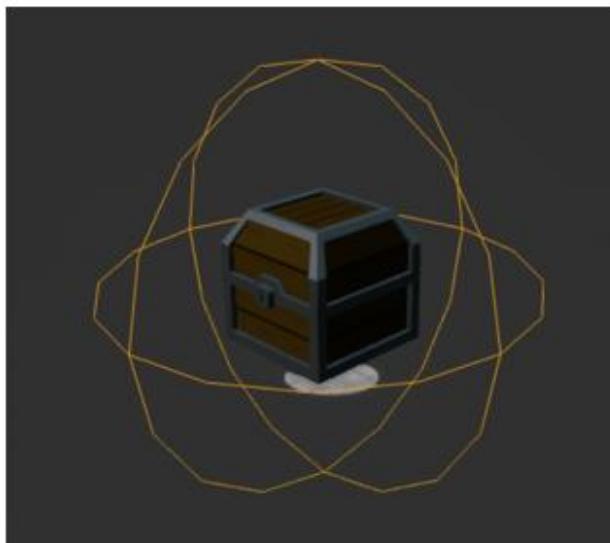
Interacțiunea cu un obiect interactibil este posibilă în momentul în care personajul se află în raza obiectului cu care acesta dorește să interacționeze. Acest lucru se poate realiza prin atașarea unei sfere/cutii/capsule de coliziune pe obiectul actual care verifică dacă personajul este sau nu în interiorul sferei pentru a începe o acțiune. Spre exemplu: un actor cufăr sau ușă. [Figură 5.14]

Actorul cufăr

Componenetele actorului

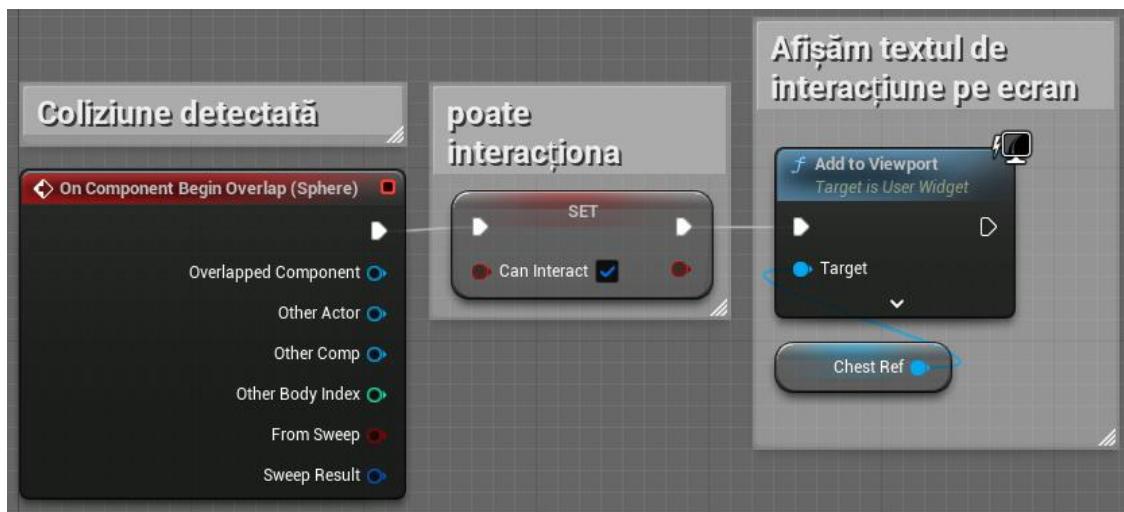


Figură 5.13 – Componențele de alcătuire a actorului cufăr



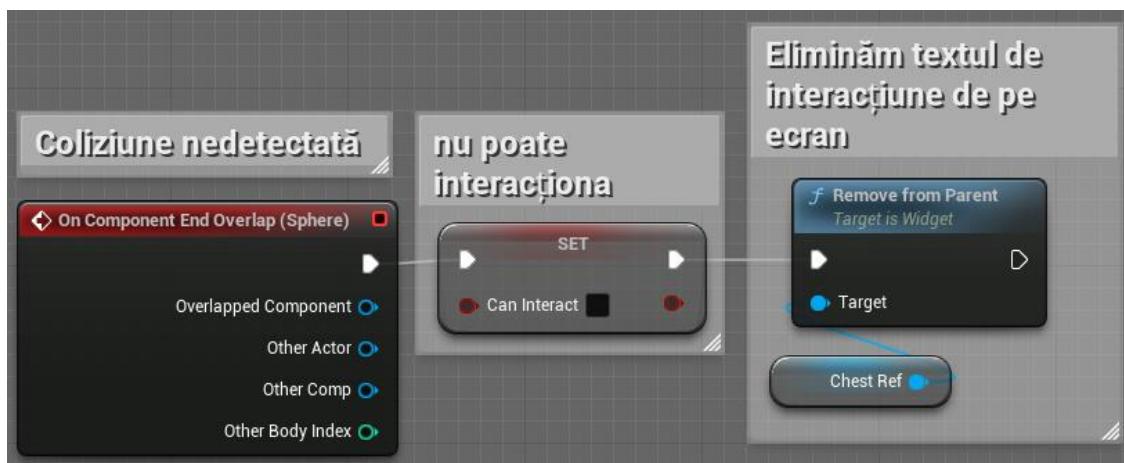
Figură 5.14 – Reprezentarea vizuală a actorului cufăr

Detectarea coliziunilor



Figură 5.15 – Detectarea coliziunilor de cufăr (Blueprint)

În momentul coliziunii a jucătorului cu sfera (Sphere)
 canInteract = True
 adăugarea textului de interacție pe ecran



Figură 5.16 – Oprirea coliziunilor de cufăr (Blueprint)

În momentul nedetectării coliziunii a jucătorului cu sfera (Sphere)
 canInteract = False
 Eliminarea textului de interacție de pe ecran

- **canInteract** – variabilă de tip boolean, utilizată în verificarea posibilității de a interacționa cu obiectul cufăr.



Figură 5.17 – Interacțiunea cu actorul cufăr (Blueprint)

În momentul în care tasta „E” este apăsată (Started)

```

IF ((chestOpened == False) AND (canInteract == True)), THEN
| Ascundem textul Open de pe ecran
| Afisăm textul Close pe ecran
| WHILE (secvența de timp nu s-a terminat), DO
| | Incrementarea în decursul unui timp stabilit a
| | fiecărui grad ( $0^\circ \rightarrow -45^\circ$ )
| | Actualizăm rotația cufărului pe axa X (în sus)
| END WHILE
| chestOpened = True
END IF

```

Interacțiunea cu acest actor se poate realiza prin următorul mecanism implementat:

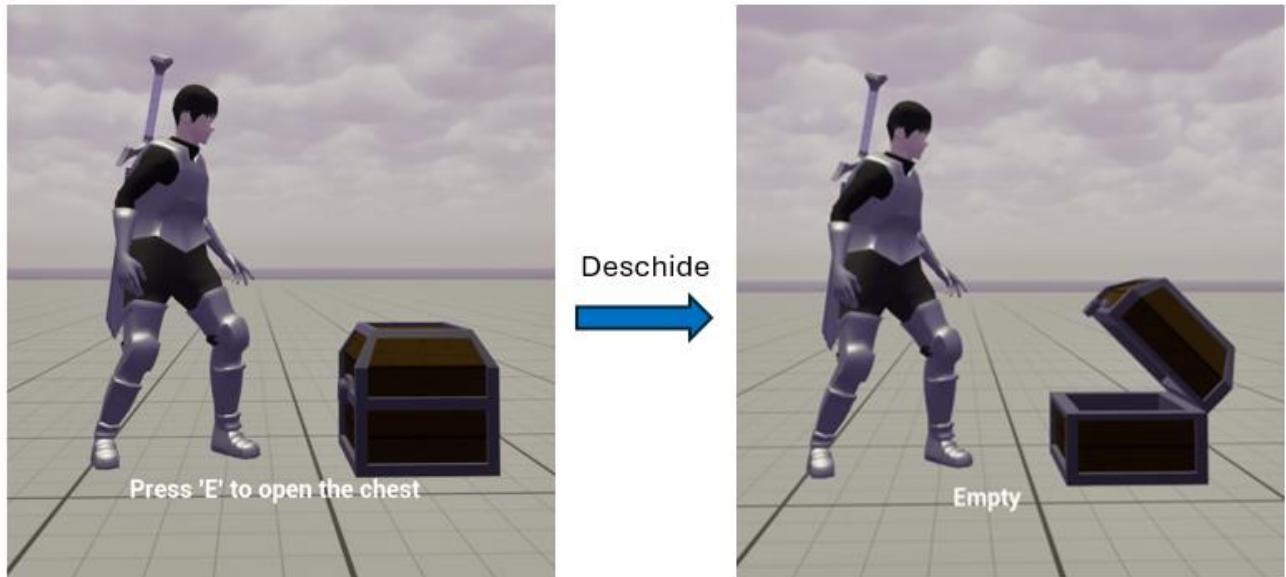
- **chestOpened**: variabilă de tip boolean, utilizată pentru a verifica dacă actorul cufăr a fost deschis sau nu. În cazul în care cufărul nu a fost deschis (chestOpened = False) putem interacționa cu acesta, iar dacă a fost deschis (chestOpened = True), atunci interacțiunea cu acesta nu va mai fi disponibilă.
- **canInteract**: variabilă de tip boolean pentru interacționarea cu actorul cufăr. În cazul în care jucătorul se află în raza de coliziune a acesteia.
- **chestRef**: referință a unui **Widget Blueprint** care afișează pe ecran textul de Open/Close.
- **Nodul Timeline**: utilizat cu scopul de a incrementa o valoare pe parcursul unui timp stabilit conectat la nodului **Lerp** cu două valori, rezultând o singura valorare în funcție de canalul alpha al acestuia pentru a seta rotația componentei actorului.

$$\text{Formula matematică pentru Lerp: } A \times (1 - \alpha) + B \times \alpha$$

Ex: $A = 0^\circ$, $B = -45^\circ$ iar $\alpha = 0.2$, atunci $0 \times (1 - 0.2) + (-45) \times 0.2 \rightarrow$ rezultat = -13.5°

- **setRelativeRotation()**: funcția este folosită pentru a seta rotația unui obiect în cazul nostru pe axa X a părții superioare a cufărului având ca și referință componenta corespunzătoare.

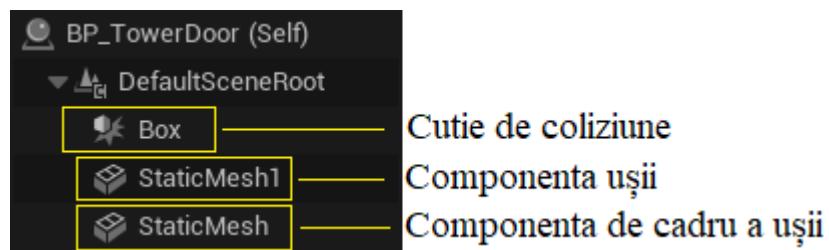
Prin urmare, acesta va fi interacțiunea cu un cufăr.



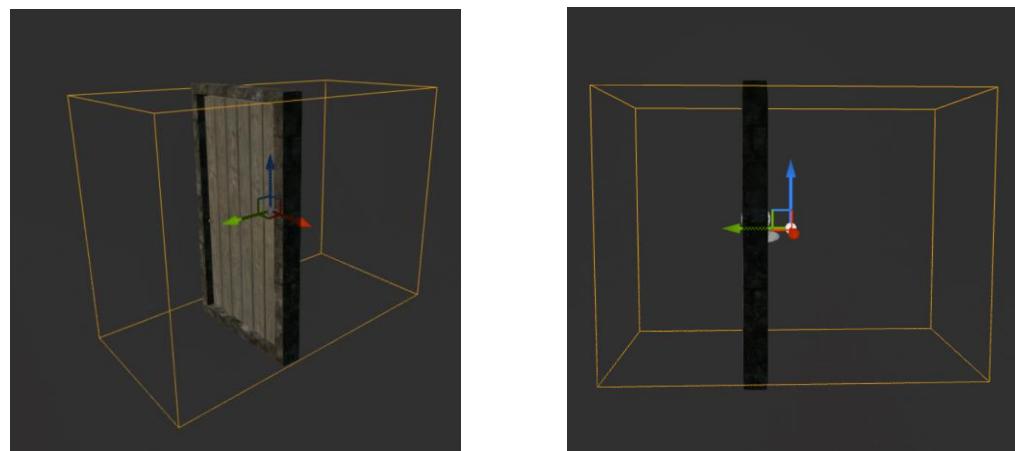
Figură 5.18 – Reprezentarea vizuală a interacțiunii cu actorul cufăr

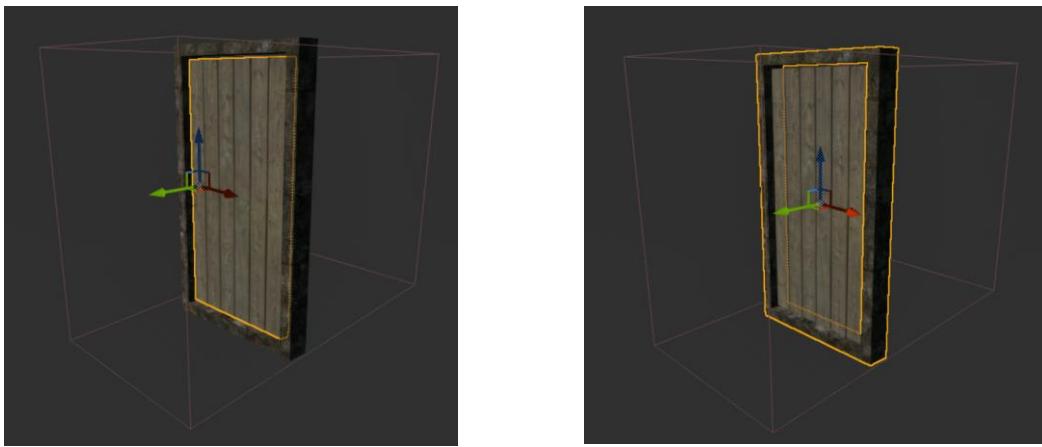
Actorul ușă

Componentele actorului



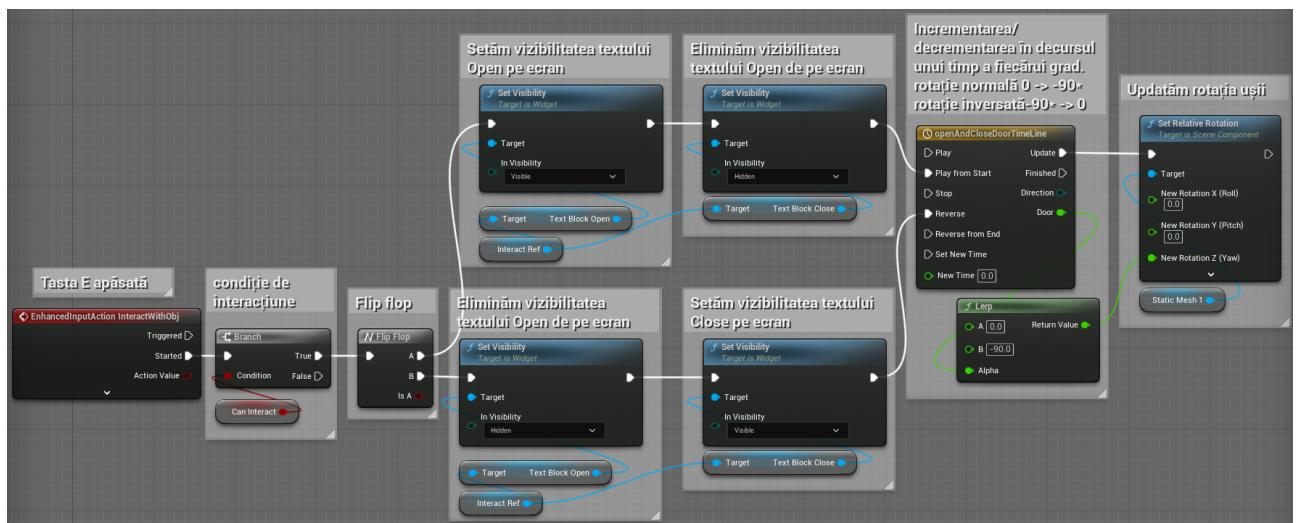
Figură 5.19 – Componentele de alcătuire ale actorului ușă





Figură 5.20 – Reprezentarea vizuală a interacțiunii cu actorul ușă

Interacțiunea cu acest actor funcționează pe același principiu de bază ca al **coliziunii actorului cufăr**, cu precizarea că **interacțiunea poate avea loc de mai multe ori**. [Figură 5.21]



Figură 5.21 – Interacțiunea cu actorul ușă (Blueprint)

Acest lucru este posibil datorită nodului **Flip-Flop**, care alternează între cele două ieșiri, A și B, la fiecare activare a intrării. Acest comportament este util în comutarea a două stări cum ar fi în cazul nostru **deschiderea sau închiderea** unei uși în mod repetat. [Figură 5.22]



Figură 5.22 – Nodul Flip-Flop

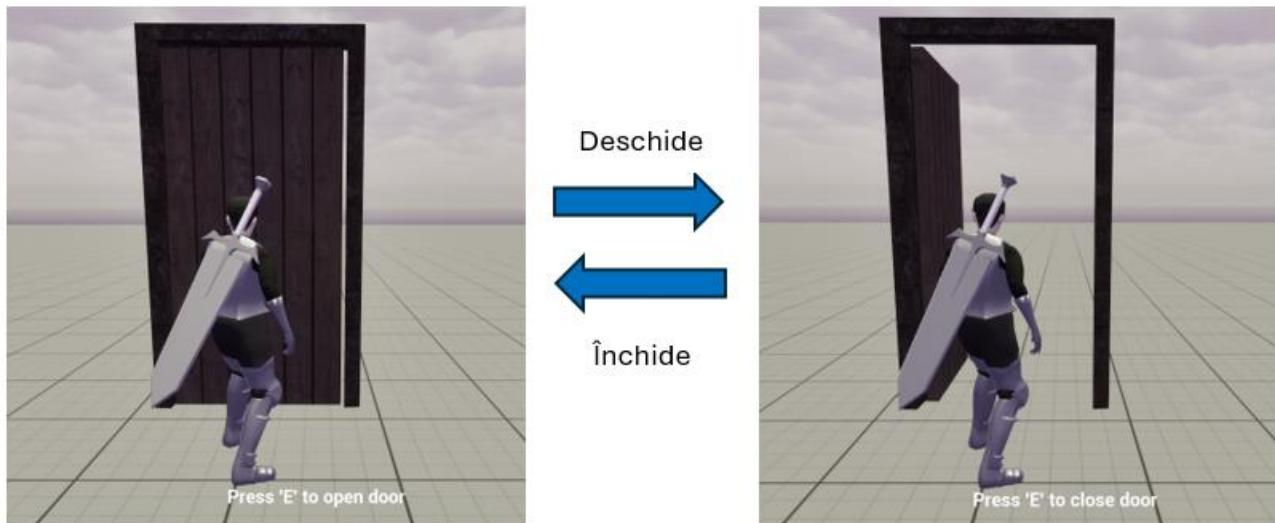
```

În momentul în care tasta „E” este apăsată (Started)
    IF (canInteract == True), THEN
        IF (Flip-Flop == A), THEN
            |   Afisăm textul Open pe ecran prin referința textului
            |   Ascundem textul Close de pe ecran prin referința acestuia
            WHILE (secvența de timp nu s-a terminat), DO
                |       Incrementarea în decursul unui timp stabilit a
                |       fiecărui grad ( $0^{\circ}$  ->  $-45^{\circ}$ )
                |       Actualizăm rotația cufărului pe axa Z (în față)
            END WHILE
        END IF

        IF (Flip-Flop == B), THEN
            |   Ascundem textul Open de pe ecran
            |   Afisăm textul Close pe ecran
            WHILE (secvența de timp nu s-a terminat), DO
                |       Reversăm în decursul unui timp stabilit a
                |       fiecărui grad ( $-45^{\circ}$  ->  $0^{\circ}$ )
                |       Actualizăm rotația cufărului pe axa Z (în spate)
            END WHILE
        END IF
    END IF

```

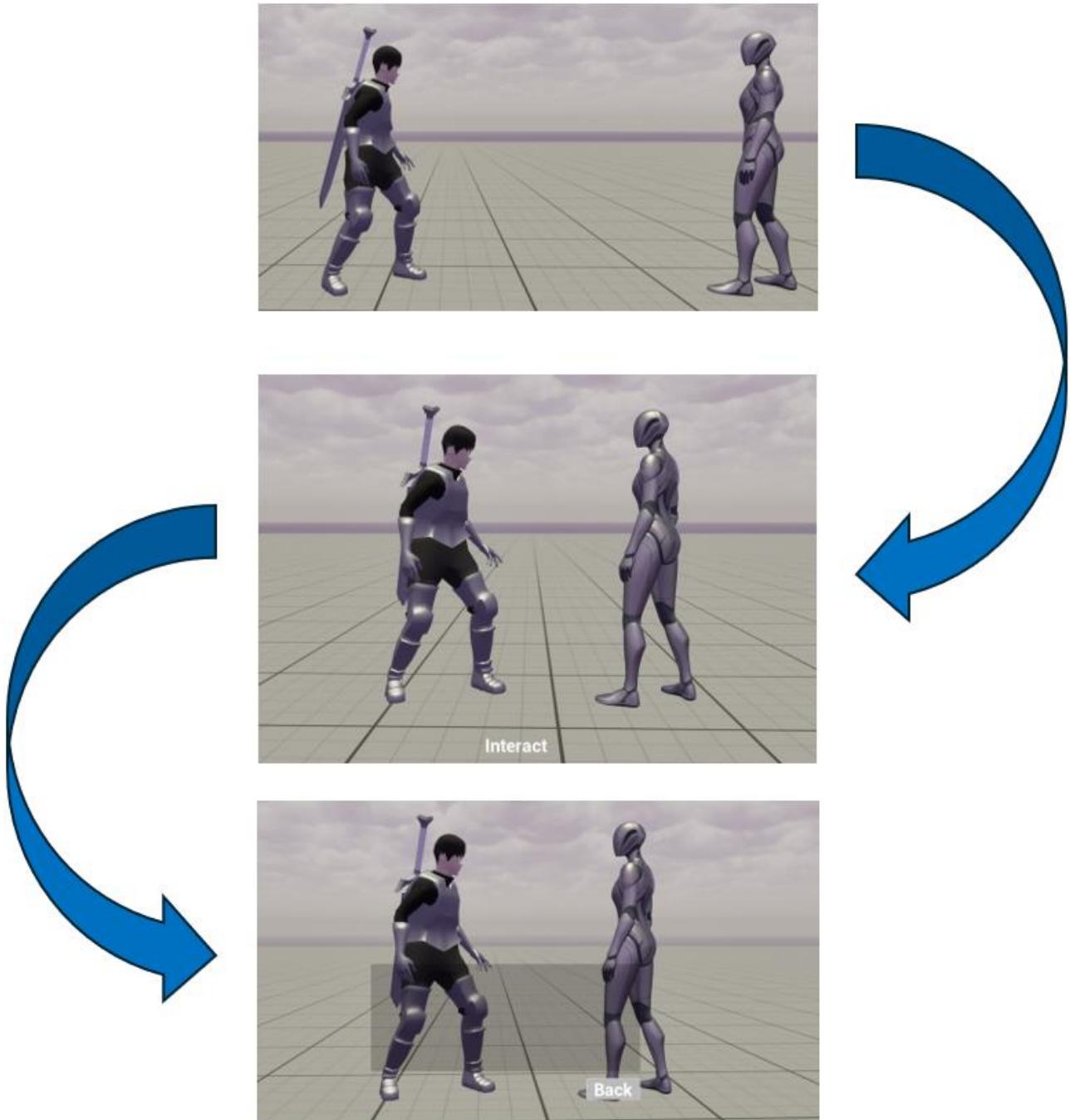
Prin urmare, acesta va fi interacțiunea cu o ușă.



Figură 5.23 – Reprezentarea vizuală a interacțiunii cu actorul ușă

Interacțiuni cu personaje prin coliziuni.

Interacțiunea cu acest actor funcționează pe același principiu de bază. Anume, dacă jucătorul se află în raza personajului (NPC), prin apăsarea tastei „E” poate interacționa cu acesta.



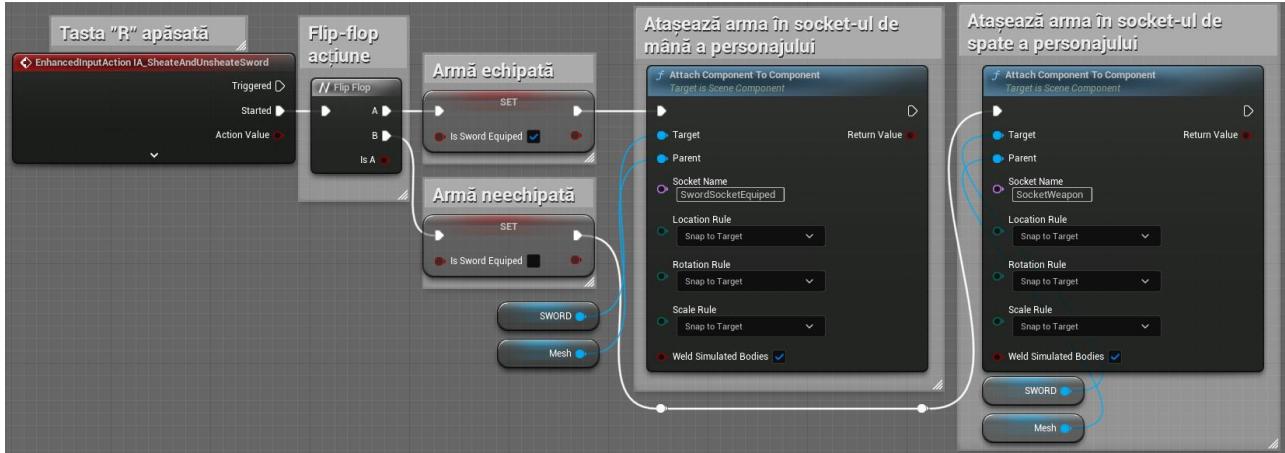
Figură 5.24 – Reprezentarea vizuală a interacțiunii cu NPC

5.2. Sub-sistemul de luptă

Sub-sistemul de luptă reprezintă în esență componenta de răspuns a unor acțiuni pentru atac sau apărare. Un sistem de luptă în general este conceput pentru ca jucătorul să se poată apăra de eventualele pericole din joc în speță oponenți (Boți AI).

Funcționalitatea de echipare și dezechipare a armei

Este un detaliu care dă contrast felului în care se simte jocul. Această funcționalitate permite echiparea armei și dezechiparea acesteia prin apăsarea tastei „R”. **O să folosi mai departe acest aspect pentru a determina posibilitatea jucătorul de a ataca sau nu în funcția condiției de echipare.** [Figură 5.25]



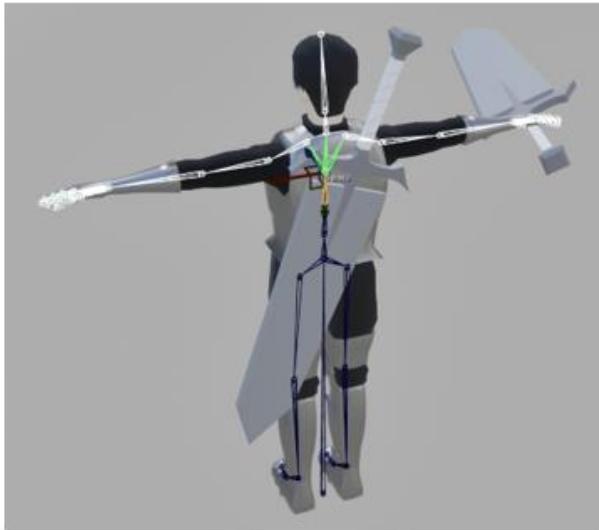
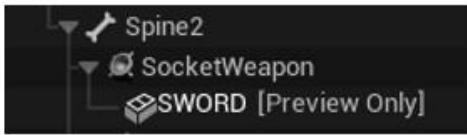
Figură 5.25 – Funcționalitatea de echipare și dezechipare a armei

```

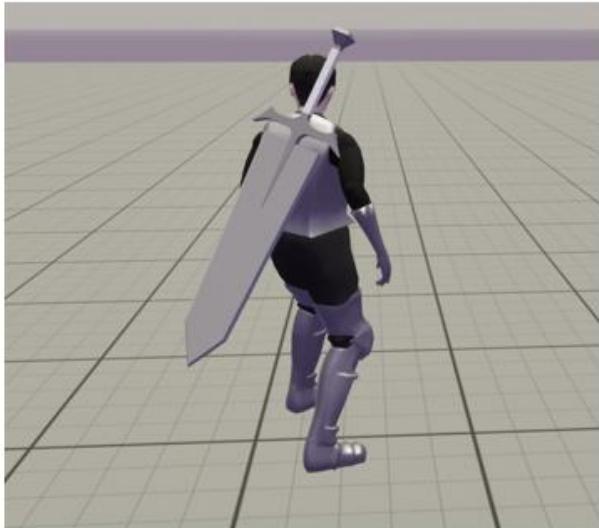
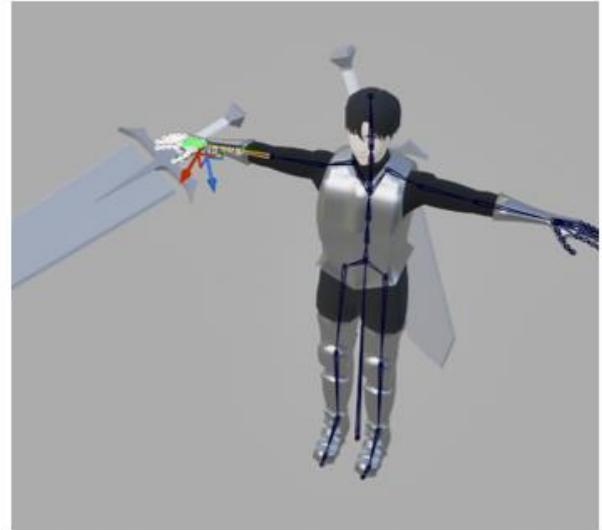
La apăsarea tastei R (Started)
IF (Flip-Flop == A), THEN
|   isSwordEquiped = True
|   Atașăm arma în socket-ul de mână al personajului
ELSE
|   isSwordEquiped = False
|   Atașăm arma în socket-ul de spate al personajului
END IF
  
```

- **isSwordEquipped:** variabilă de tip boolean, care validează echiparea de armă.
- **attachComponentToComponent():** funcția atașează o componentă la altă componentă într-un socket creat pe scheletul personajului. [28]

Socket-ul armei de pe spate



Socket-ul armei din mâna

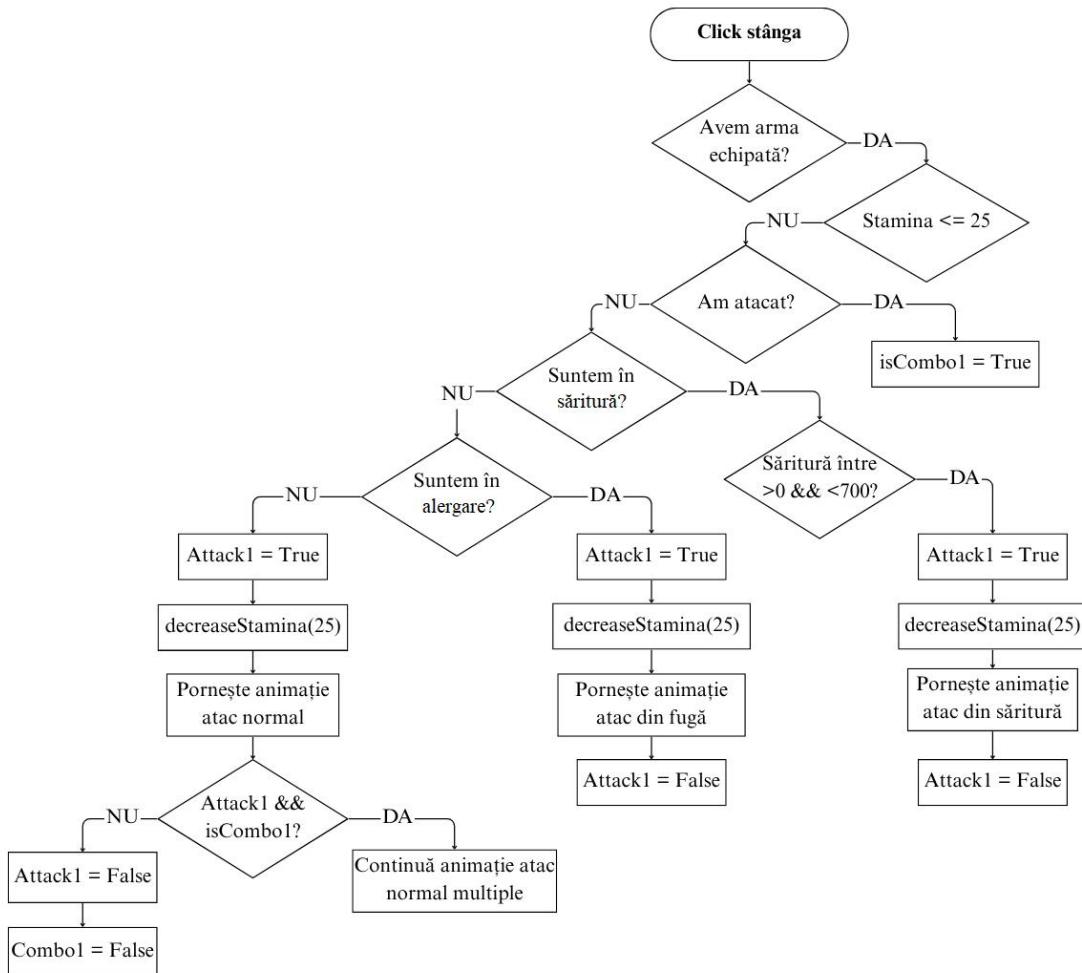


Figură 5.26 – Reprezentarea vizuală de echipare și dezechipare a armei

5.2.1. Atacuri contextuale

Atacurile contextuale sunt acțiuni de atac care sunt diferențiate una față de cealaltă, prin condiții sau stări ale personajului. Atacurile contextuale sunt esențiale unui joc pentru a oferi jucătorului o experiență placută prin posibilitățile multiple de acțiune a atacurilor pe care i le sunt oferite.

Organograma atacurilor contextuale ușoare (Light attacks)

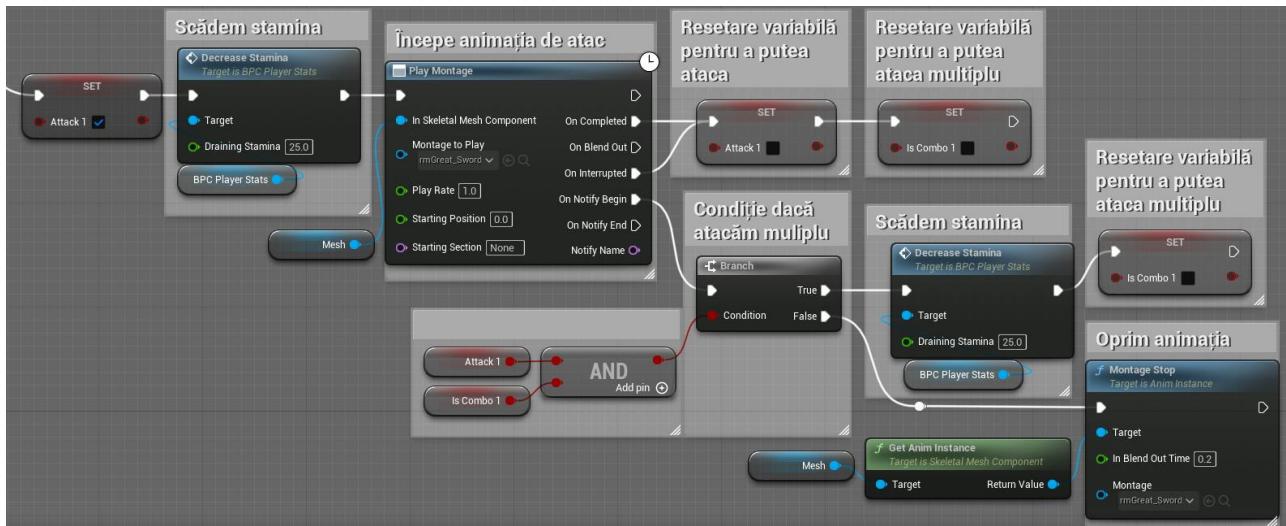


Figură 5.27 – Organograma atacurilor contextuale ușoare

Organigramă reprezintă modul și succesiunea de instrucțiuni pe care le urmează pentru alegerea unui anumit tip de atac. [Figură 5.27] Organigramă este alcătuită din următoarele tipuri de atacuri:

- Atac normal/multiple
- Atac din alergare
- Atac din săritură
- Atac din evitare

Atacul normal/multiplu (Basic/multiple attack/s)

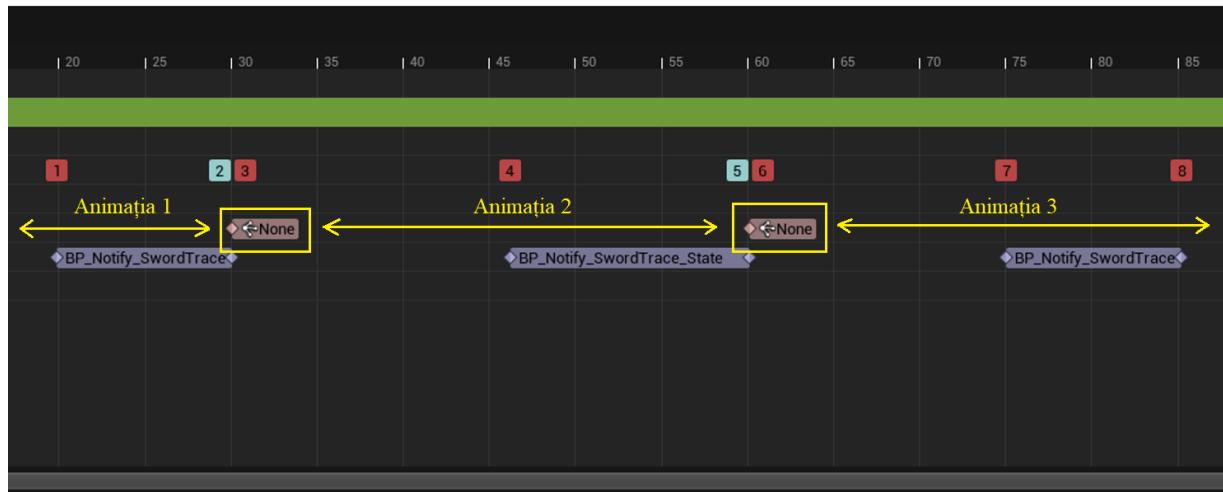


Figură 5.28 – Atacul normal/multiplu

Prin apăsăm click-ului stânga de la mouse, vom ataca cu arma iar animația de atac v-a începe să ruleze.

onNotifyBegin – este un output al **Animațiilor de monaj** folosit pentru a verifica un eveniment la un moment dat din durata animației. [25]

Animația de atac normal, pe care o folosim, este alcătuită din trei mișcari puse cap la cap. Însă pentru a le putea accesa în parte, avem nevoie de ceva prin care să le separăm. Soluția ar fi ca pe fiecare atac să îl marcăm cu o notificare, pentru a le putea diferenția. **[Figură 5.28]**



Figură 5.29 – Marcarea punctelor în secvența animațiilor de monaj

În acest fel am separat o animație în trei secvențe folosind doar două notificări.

Tip de atac	Animații folosite	Număr click-uri stânga
Atac normal	Animația 1 (1 singur atac)	1 click
Atac multiplu	Animația 1 urmată de Animația 2 (2 atacuri)	2 click-uri
Atac multiplu	Animația 1 și 2 urmată de Animația 3 (3 atacuri)	3 click-uri

Tabel 5.3 – Tabelul atacurilor ușoare

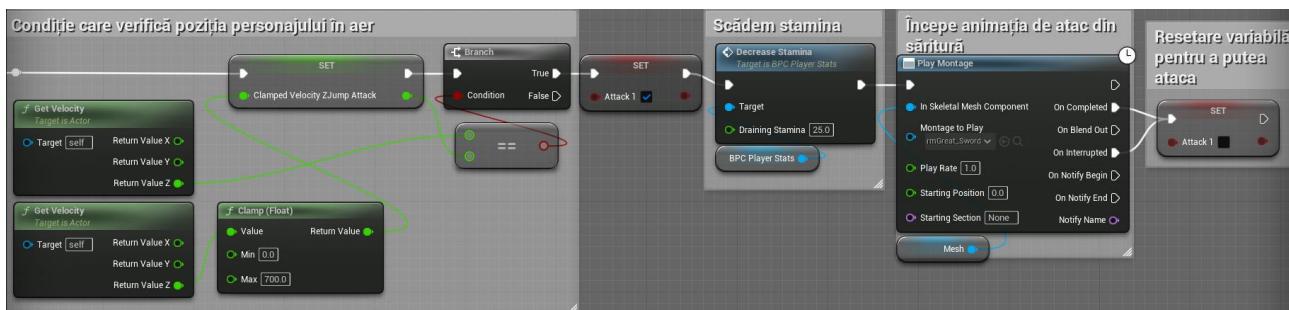
Atacul din alergare (Sprint attack)



Figură 5.30 – Atacul din alergare (Blueprint)

Prin tinerea apăsată a tastei „Shift” sau mai precis când personajul este **în starea de alergat**, în combinație cu butonul click stânga al mouse-ului se va realiza astfel, un alt atac contextual condiționat de variabila **isSprinting** din cadrul **Functionalității de alergare**. [Figură 5.30]

Atacul din săritură (Jump attack)



Figură 5.31 – Atacul din săritură (Blueprint)

Prin apăsarea tastei „Space” sau mai precis când personajul se află **în starea de săritură**, în combinație cu butonul click stânga al mouse-ului se va realiza astfel, un alt atac contextual condiționat de variabila **isJumping** din cadrul **Functionalității de săritură**. [Figură 5.31]

Atacul din evitare (Dodge attack)

Acest tip de atac se încadrează în organograma atacurilor contextuale, prin apasă tastei „CTRL” (animația de evitare) + click stânga (atac normal).

La apăsarea tastei „CTRL” (Started)
 Pornim animația de evitare
 decreaseStamina(25)
 Atacăm (Dodge Attack)

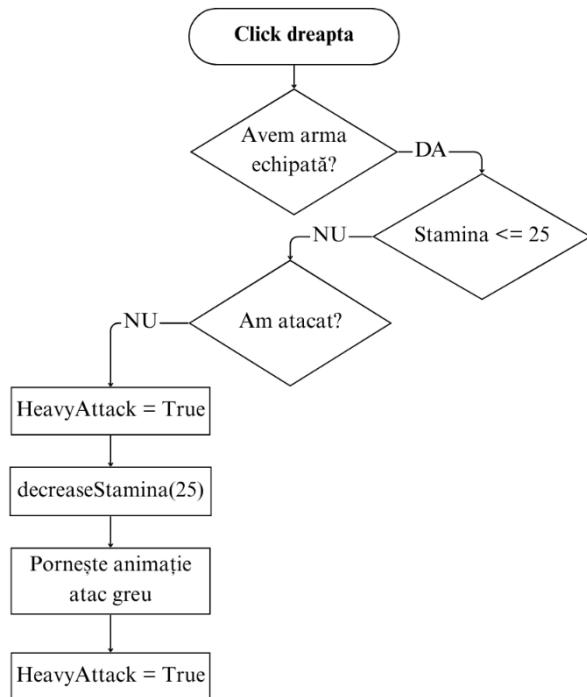
```

La apăsarea click dreapta (Started)
    IF(isSwordEquipped == True), THEN
        IF((Stamina <= 25) == False), THEN
            | | IF(Attack1 == True), THEN
            | | | isCombo1 = True
            | | ELSE
            | | | IF(isJumping == True), THEN
            | | | | Setăm velocitatea pe axa Z,
            | | | | clamp(min(0):max(700))
            | | | | IF((velocitatea >= 0 && <=700) == True), THEN
            | | | | | Attack1 = True
            | | | | | decreaseStamina(25)
            | | | | | Începem animația de atac din săritură
            | | | | | (Jump Attack)
            | | | | | Attack1 = False
            | | | END IF
            | | ELSE
            | | | IF(isSprinting == True), THEN
            | | | | Attack1 = True
            | | | | decreaseStamina(25)
            | | | | Începem animația de atac din alergare
            | | | | (Sprint Attack)
            | | | | Attack1 = False
            | | ELSE
            | | | | decreaseStamina(25)
            | | | | Începe animația de atac normal
            | | | | (Basic/Multiple Attack/s)
            | | | | IF(animația s-a terminat? == True), THEN
            | | | | | Attack1 = False
            | | | | | Combo1 = False
            | | | ELSE
            | | | | | IF(Attack1 && Combo1 == True), THEN
            | | | | | | decreaseStamina(25)
            | | | | | | isCombo1 = False
            | | | | ELSE
            | | | | | | Oprim Animația
            | | | | END IF
            | | | END IF
        END IF
    END IF
END IF

```

- **isSwordEquipped**: variabilă de tip boolean care verifică dacă personajul are arma echipată sau nu pentru a face posibil atacul.
- **velocitatea**: valoare de validare a momentului în care personajul se află în starea de săritură. Dacă ne încadrăm între cei doi parametrii de adevăr atunci va fi posibil atacul din săritură.
- **attack1**: variabilă de tip boolean care indică dacă jucătorul a pornit un atac. Prin resetarea acesteia jucătorul va putea ataca din nou.
- **isCombo1**: variabilă de tip boolean utilizată pentru validarea posibilității de atac multiplu pe parcursul animației.

Organograma atacului greu (Heavy attack)



Figură 5.32 – Organograma atacului greu

Organograma reprezintă modul și succesiunea de instrucțiuni pe care le urmează pentru a realiza acest tip de atac.



Figură 5.33 – Atacul greu (Blueprint)

La apăsarea butonului **click dreapta** al mouse-ului se va executa atacul greu (heavy). Iar prin combinarea cu celelalte contexte vor rezulta alte atacuri adiționale sub-sistemului nostru de atac.

```

Prin apăsarea butonului click dreapta de la mouse (Started)
IF(isSwordEquipped == True), THEN
|   IF((Stamina <= 25) == False), THEN
|   |   IF (heavyAttack == False), THEN
|   |   |   heavyAttack = True
|   |   |   decreaseStamina(25)
|   |   |   Începe animația de atac greu (Heavy Attack)
|   |   END IF
|   END IF
END IF

```

- **heavyAttack**: variabilă de tip boolean care verifică dacă jucătorul atacă. Resetarea ei va face posibilă condiția de atac nou.

Tabelul atacurilor contextuale

Tabelul atacurilor contextuale oferă o privire de ansamblu al combinațiile de taste pe care jucătorul le poate apăsa pentru a îndeplini un anumit tip de atac.

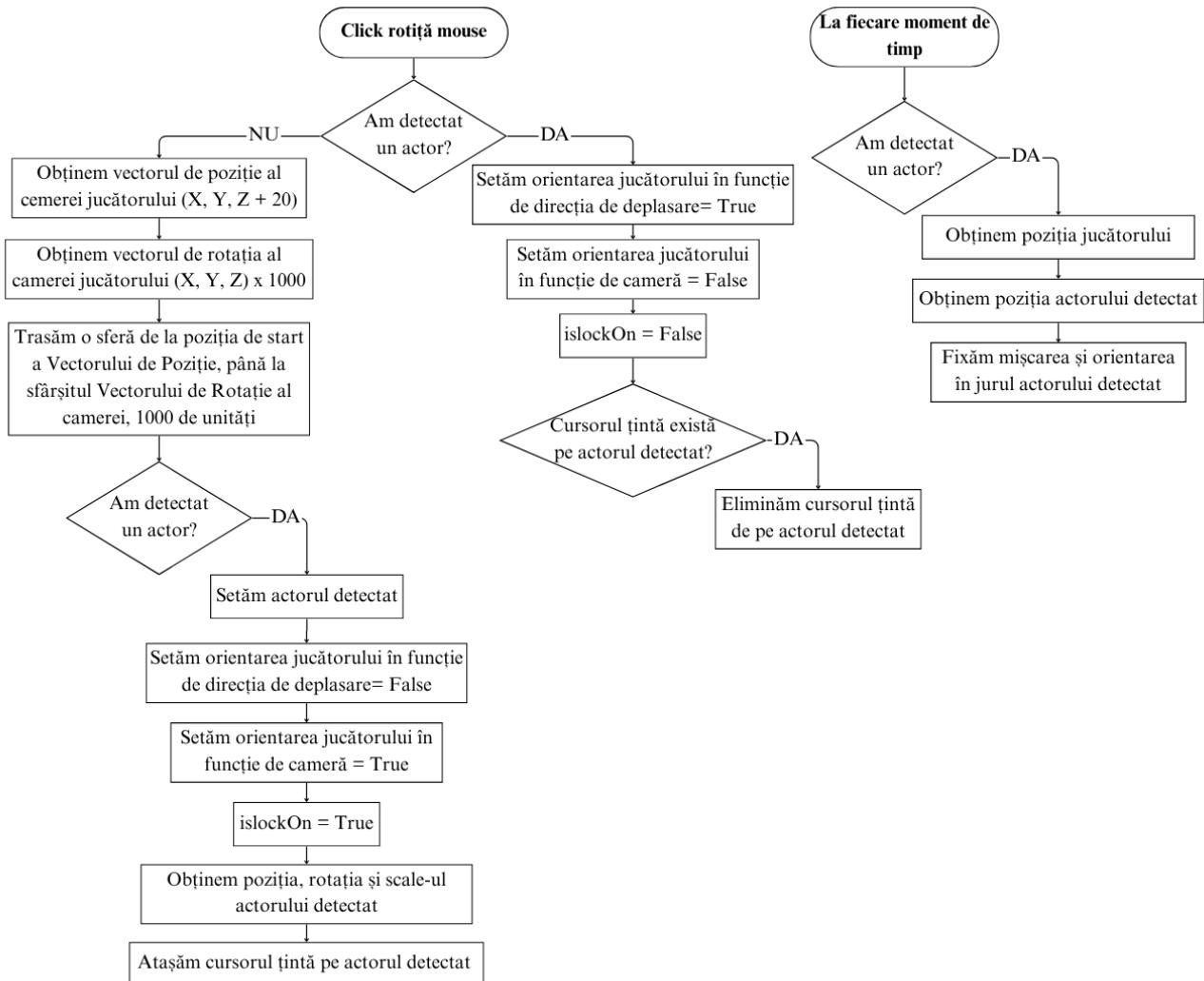
NR.	Taste apăsate	Tipul de context	Apartenenta atacului
1	Click stânga	Atac normal (Basic attack)	Light attack
2	2 x click stânga	Atac multiplu (Multiple attack)	Light attack
3	3 x click stânga	Atac multiplu (Multiple attack)	Light attack
4	Space + click stânga	Atac din săritură (Jump attack)	Light attack
5	Shift + click stânga	Atac din alergare (Spring attack)	Light attack
6	Ctrl + click stânga	Atac din evitare (Dodge attack)	Light attack
7	Click dreapta	Atac normal (Basic attack)	Heavy attack
8	Space + click dreapta	Atac din săritură (Jump attack)	Heavy attack
9	Shift + click dreapta	Atac din alergare (Spring attack)	Heavy attack
10	Ctrl + click dreapta	Atac din evitare (Dodge attack)	Heavy attack

Tabel 5.4 – Tabelul atacurilor contextuale

5.2.2. Lock-on

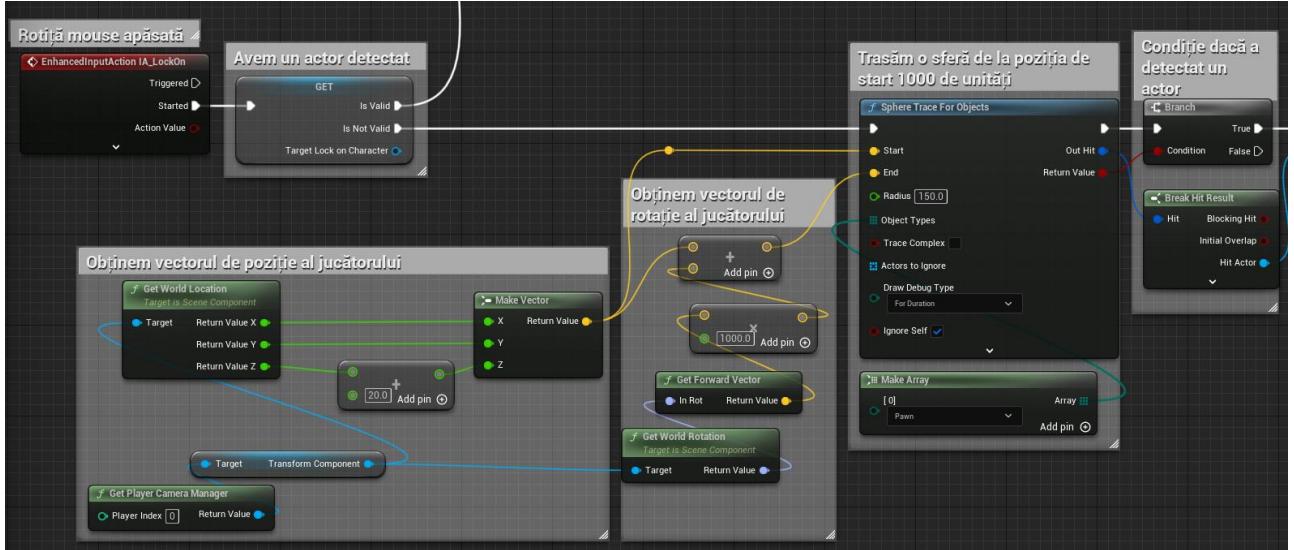
Cuvântul **lock-on** reprezintă **mecanismul prin care jucătorul poate fixa camera și direcția de atac** asupra unui oponent, AI sau chiar obiect. Această funcționalitate este frecvent întâlnită în jocurile video care **ajută la urmărirea mai ușoară a țintelor și preciziei de atac**.

Organigramă generală Lock-on



Figură 5.34 – Organigramă generală Lock-on

Organigramă reprezintă modul și succesiunea de instrucțiuni pe care le urmează pentru ca mișcarea și orientarea camerei să fie fixată asupra unei ținte.



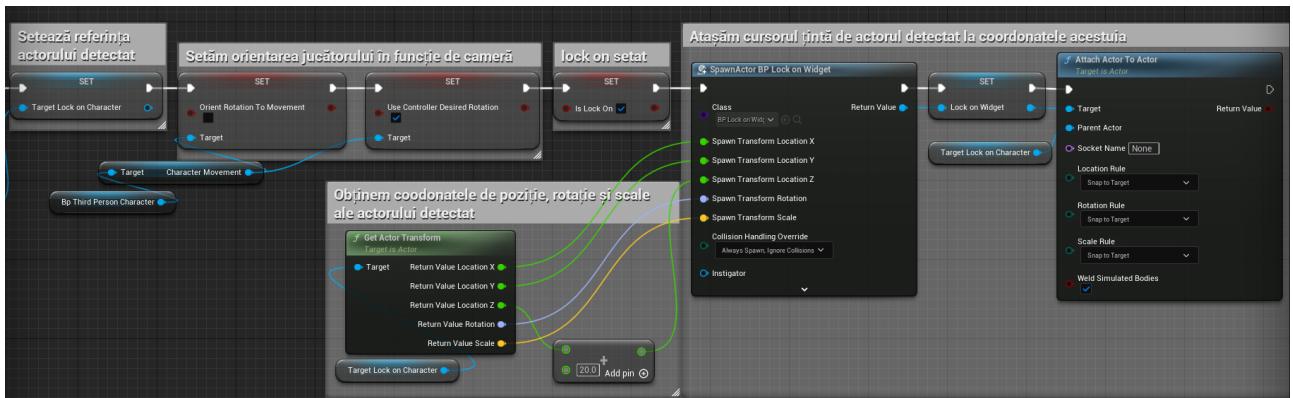
Figură 5.35 – Funcționalitatea Lock-on (Blueprint)

Pentru a detecta un actor, la apăsarea pe rotiță a mouse-ului, trasăm o sferă de la poziția de start a camerei, către sfârșit, 1000 de unități în direcția în care avem îndreptată camera. Trasarea de sferă v-a declanșa o coliziune în momentul suprapunerii cu actorii de tip Pawn. [Figură 5.36]



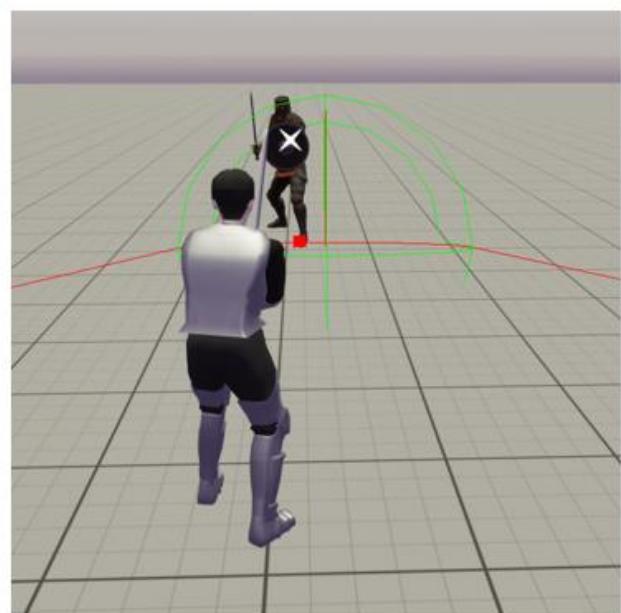
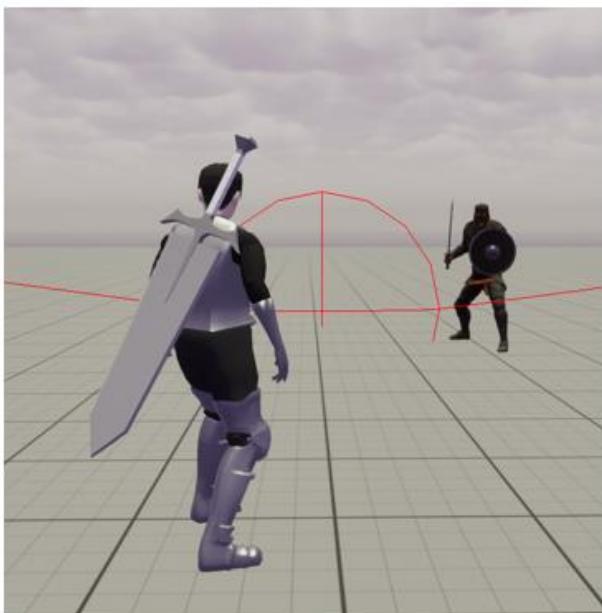
Figură 5.36 – Trasarea sferei de detectare a unui actor

Nodul **Break Hit Result**, returnează condiția de coliziune a sferei cu actorul și setarea referinței actorului detectat.



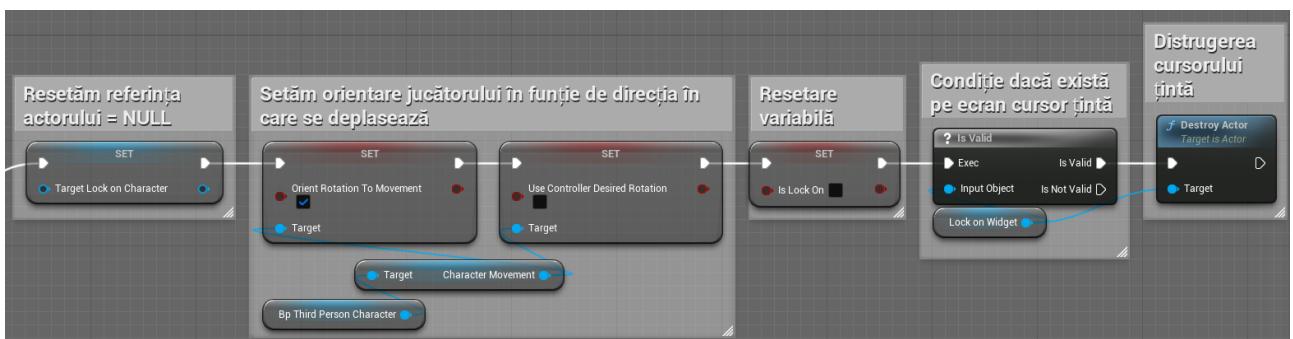
Figură 5.37 – Fixare cursor Lock-on pe actor (Blueprint)

Dacă condiția de coliziune a fost îndeplinită, atunci setăm orientarea jucătorului către actorul detectat și în același timp atașarea unui cursor țintă pentru a-l putea urmări. [Figură 3.38]



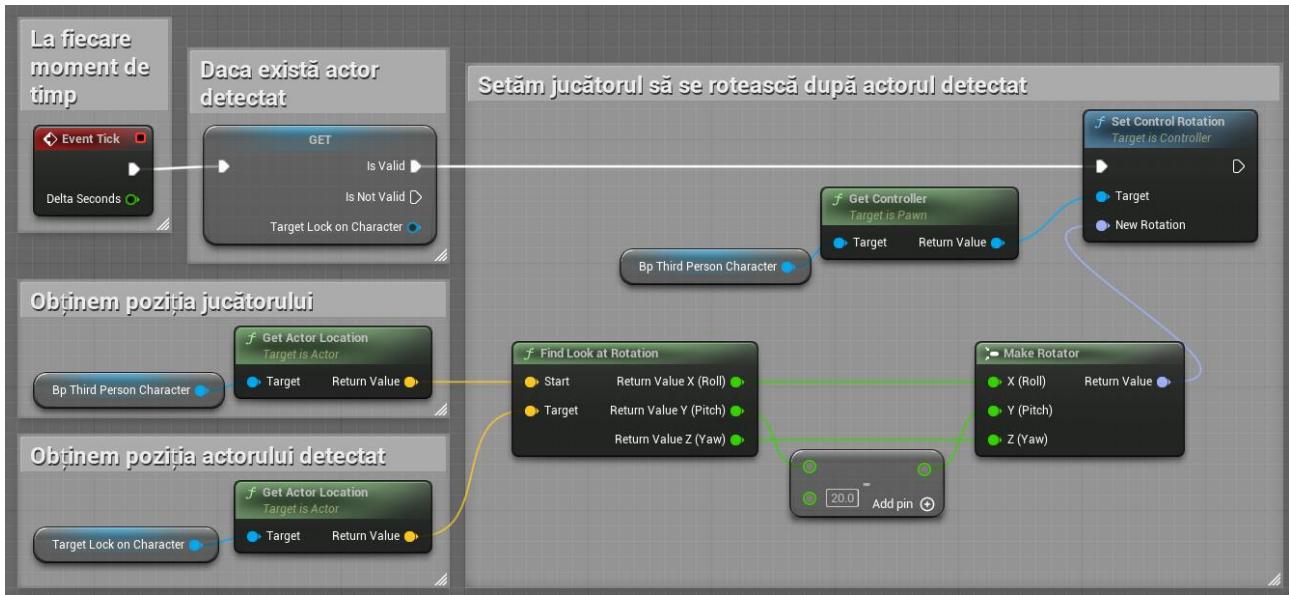
Figură 5.38 – Reprezentarea vizuală a modului Lock-on

De asemenea putem reveni la controlul default prin simpla apăsare a rotitei mouse-ului care resetează referința actorului detectat la NULL, modului în care ne mișcăm și eliminarea cursorului țintă de pe ecran. [Figură 5.39]



Figură 5.39 – Eliminarea cursorului de pe ecran (Blueprint)

Actualizarea constantă a orientării camerei este realizată de următorul mecanism, atât timp cât actorul detectat este valid setăm orientarea în direcția acestuia pentru a îl urmări.



Figură 5.40 – Actualizarea camerei după oponent Lock-on (Blueprint)

```

La apăsarea rotitei mouse-ului (Started)
IF(actor detectat == NULL), THEN
| Obținem vectorul de poziție al jucătorului
| Obținem vectorul de rotație al jucătorului (în față) x 1000
| Trasăm sferă de coliziune în direcția în care avem
| îndreptată camera
| IF(actor detectat == Pawn[i]), THEN
| | targetLockOnCharacter = Pawn[i]
| | Orientare în funcție de mișcare = False
| | Orientare în funcție de rotația camerei = True
| | isLockOn = True
| | Creăm actorul cursor țintă
| | Atașăm cursorul țintă de actorul detectat
| END IF
ELSE
| targetLockOnCharacter = NULL
| Orientarea în funcție de mișcare = True
| Orientarea în funcție de rotația camerei = False
| isLockOn = False
| IF(cursorul țintă există), THEN
| | Eliminăm cursorul țintă
| END IF
END IF

La fiecare moment de timp(cadrul)
IF(actor detectat != NULL), THEN
| Obținem poziția jucătorului
| Obținem poziția actorului detectat
| Setăm camera jucătorului fixată pe actorul detectat
END IF

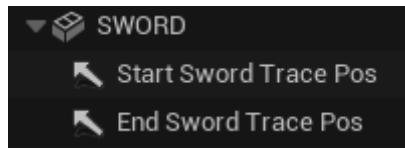
```

5.2.3. Trasări de linii pentru atacuri

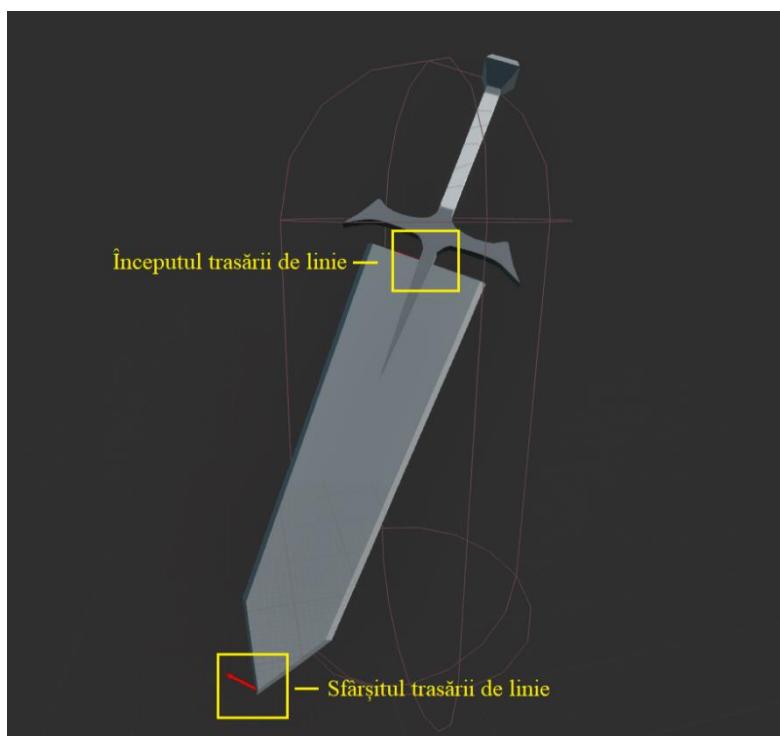
Trasările de linii (Line Traces) sunt utilizate în contextul **animațiilor de montaj** ale atacurilor. Acestea sunt esențiale pentru detecția unui eveniment de coliziunie în momentul atacurilor cu un obiect sau actor prin intermediul **notificărilor** plasate la secvențe de timp diferite care determină începutul și sfârșitul de desen al liniilor.

Prin urmare obiectul de armă al jucătorului va fi compus din trei elemente:

- Modelul de armă (mesh)
- componentă săgeată care indică începutul trasării de linie
- componentă săgeată care indică sfârșitul trasării de linie



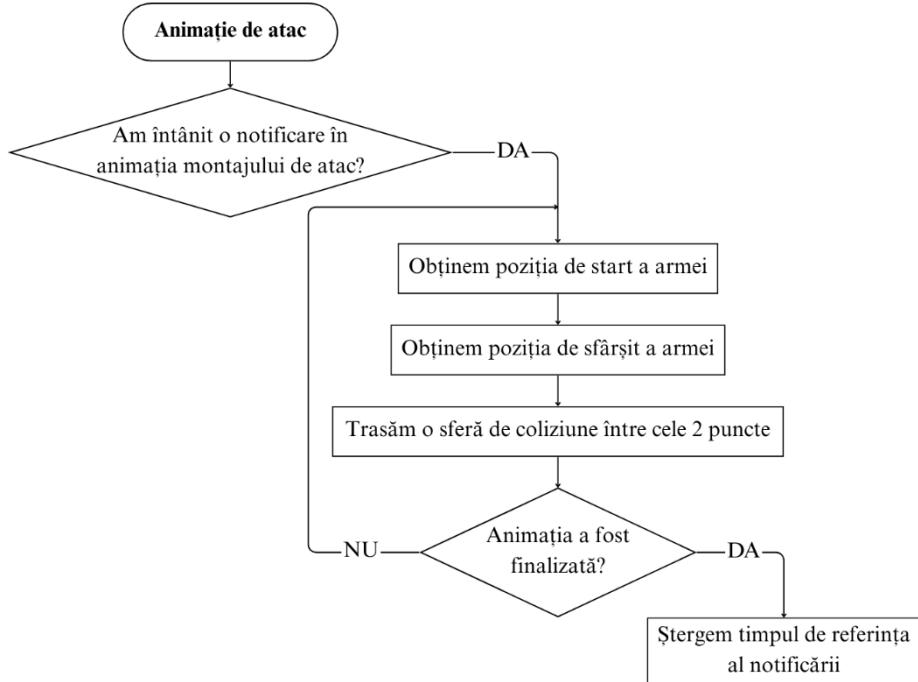
Figură 5.42 – Componentele de alcătuire a punctelor de început și sfârșit trasare linie



Figură 5.41 – Reprezentarea vizuală a punctelor de început și sfârșit trasare linie

Cele două săgeți sunt importante deoarece vor determina poziția de start și sfârșit a desenării liniilor pe ecran pentru a facilita coliziunile de atac cu actorii. [Figură 5.42]

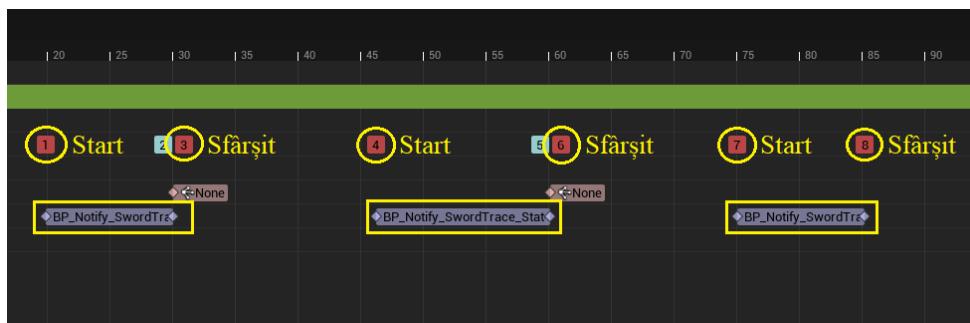
Organograma generală a trasării de linii pentru atacuri



Figură 5.43 – Organograma generală a trasării de linie pentru atacuri

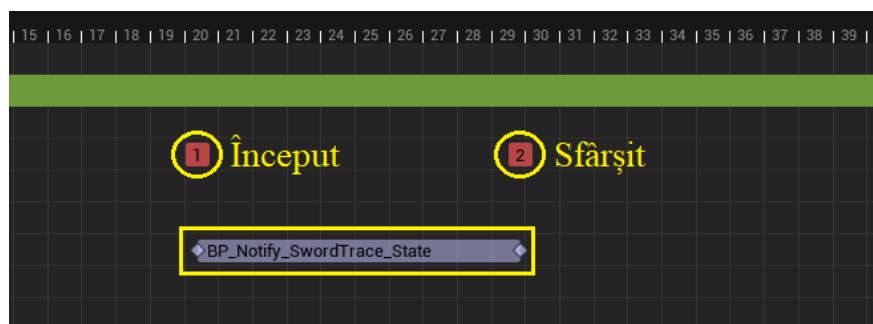
Exemple de marcarea notificărilor:

Utilizat pentru atacul normal/multiplu (Normal/Multiple attack)



Figură 5.44 – Marcarea notificărilor în animația de montaj pentru atac normal/multiplu

Utilizat pentru atacul greu (Heavy attack)

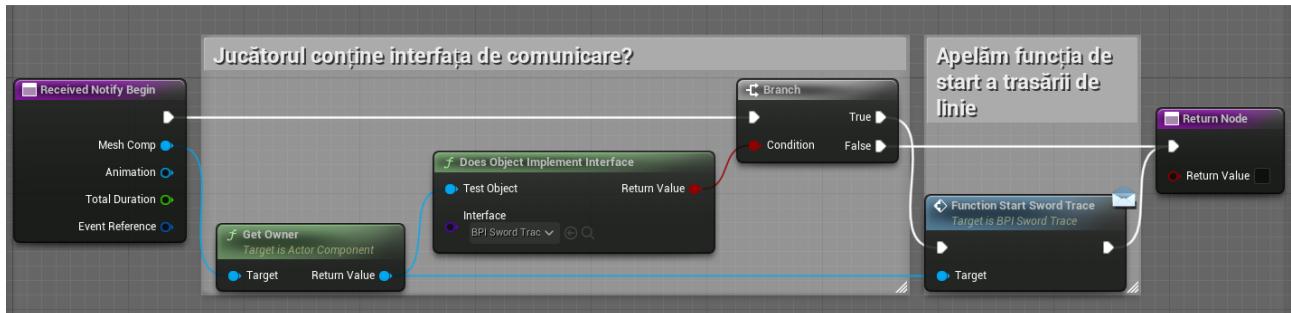


Figură 5.45 – Marcarea notificărilor în animația de montaj pentru atac greu

Notificarea constă din două funcții necesare:

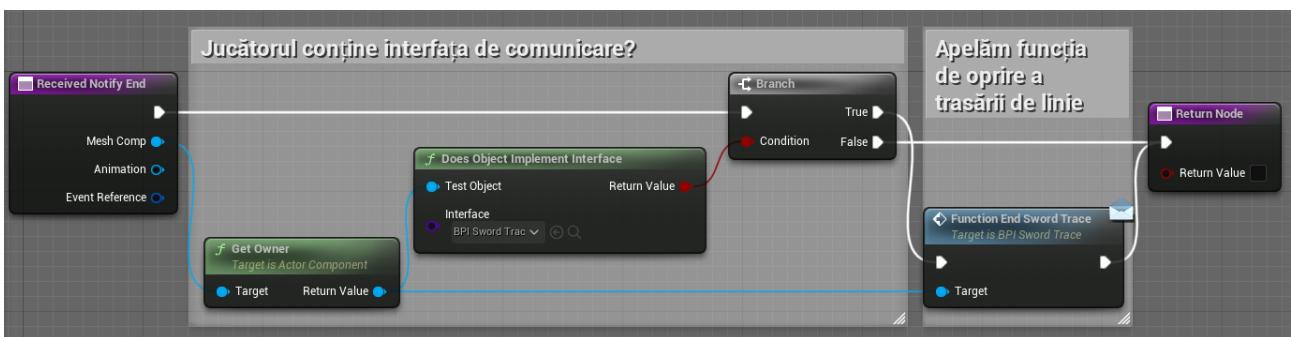
- Primirea de notificare
- Sfârșitul de notificare

Primirea de notificare (Receive notify begin) verifică dacă jucătorul implementează interfața blueprint prin care comunică cu funcțiile specifice acesteia. Dacă există, atunci apelăm funcția responsabilă cu începerea trasării de linie, `startSwordTrace()`. [Figură 5.46]



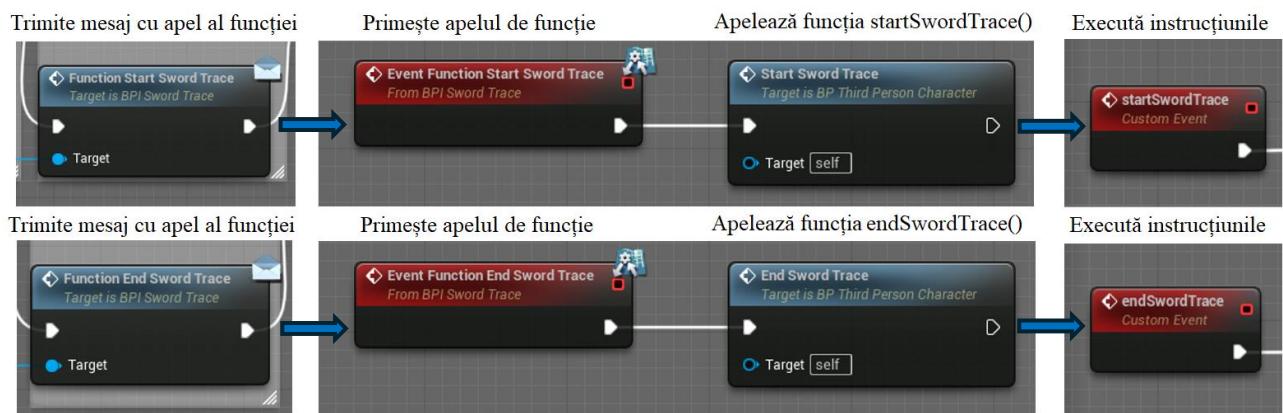
Figură 5.46 – Primirea de notificare (Blueprint)

Sfârșitul de notificare (Receive notify end) verifică dacă jucătorul implementează interfața blueprint prin care comunică cu funcțiile specifice acesteia. Dacă există, atunci apelăm funcția responsabilă cu sfârșitul trasării de linie, `endSwordTrace()`. [Figură 5.47]

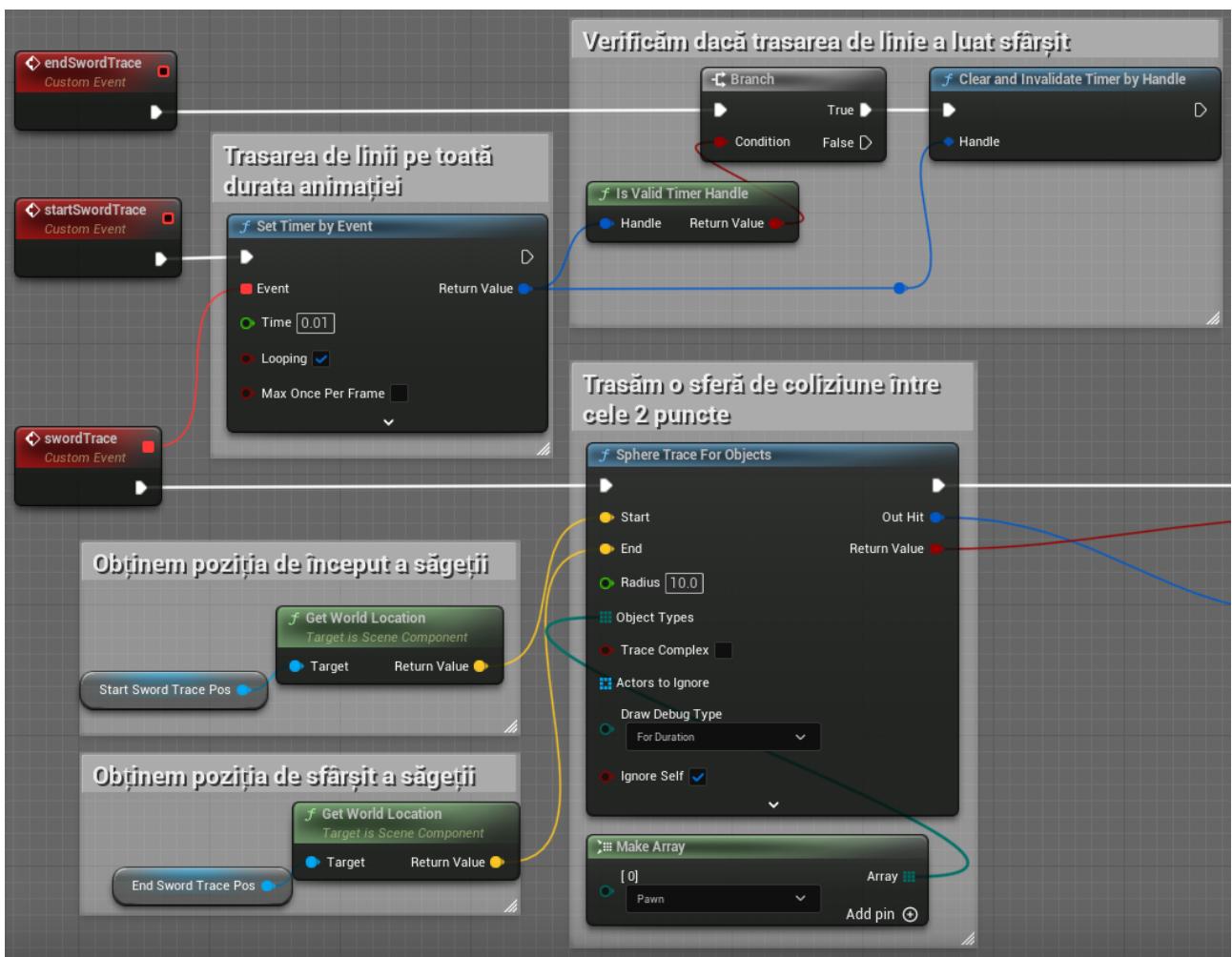


Figură 5.47 – Sfârșitul de notificare (Blueprint)

Interfața prin care comunică va fi una de **Blueprint Interface**, permitând apelarea funcțiilor prin comunicare între cele două blueprint-uri, **BPI Sword Trace** și **Jucător**, în următorul mod. [Figură 5.48]



Figură 5.48 – Modul de comunicare între BPISwordTrace și jucător



Figură 5.49 – Trasarea de linii (Blueprint)

În momentul atacului

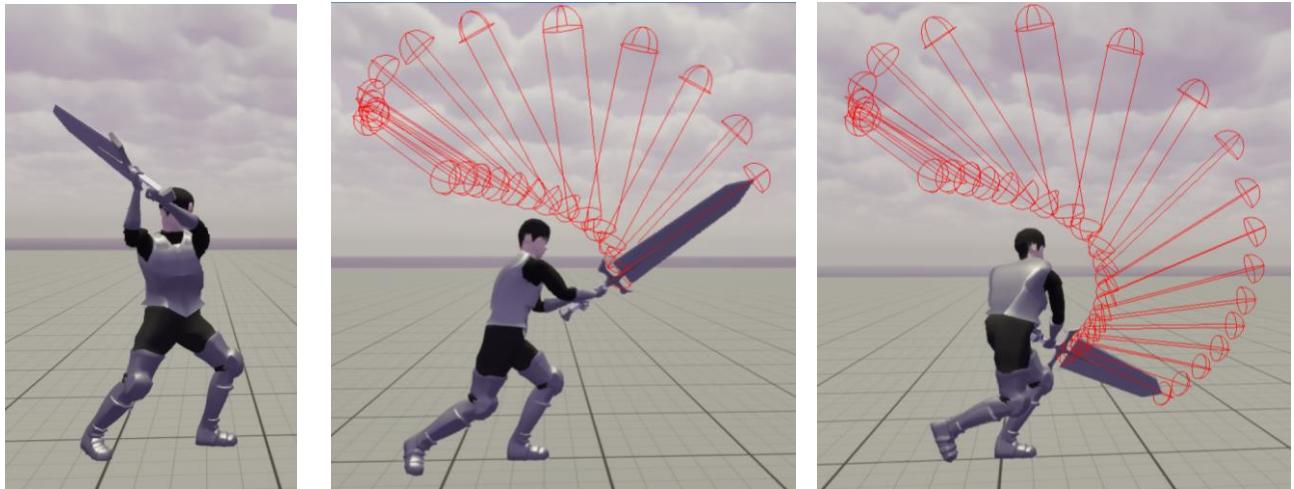
```

IF (Am întâlnit o notificare în animația de montaj == True), THEN
|   Executăm primirea de notificare
|   IF (jucătorul are interfață de comunicare == True), THEN
|   |   DO
|   |   |   Obținem poziția de start a armei
|   |   |   Obținem poziția de sfârșit a armei
|   |   |   Trasăm o sferă de coliziune între cele 2 puncte
|   |   WHILE (notificare încă activă (animația a fost
completată?))
|   |   Executăm sfârșitul de notificare
|   |   IF (jucătorul are interfață de comunicare == True), THEN
|   |   |   Resetăm notificarea
|   |   END IF
|   END IF
END IF

```

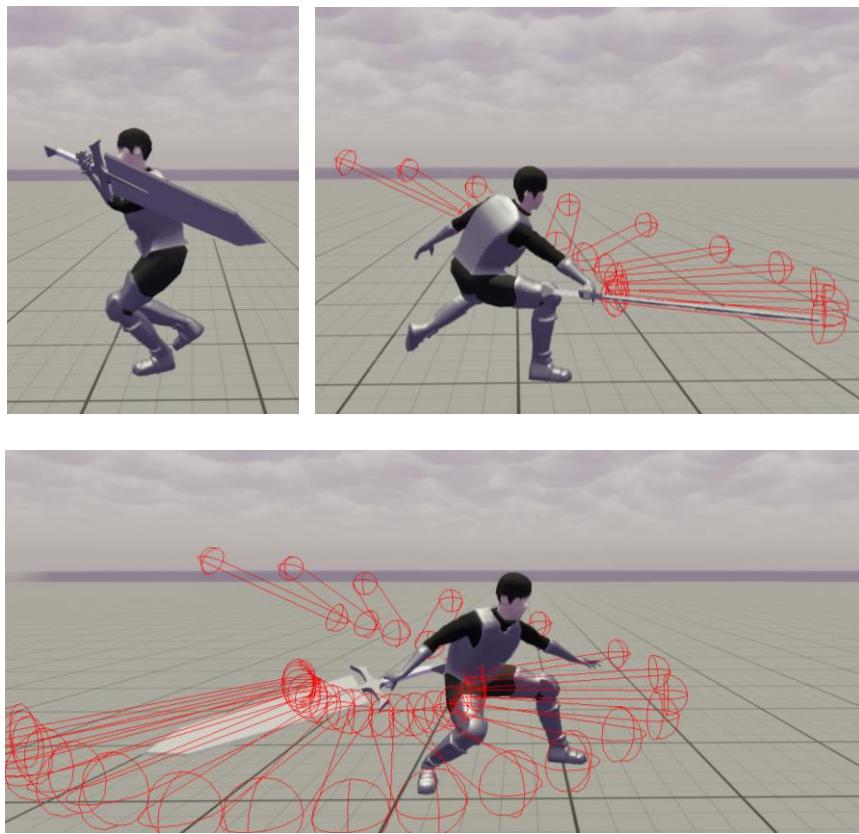
Ca exemplu, acesta va fi rezultatul trasării de linii în momentul atacurilor.

Atacului normal (Basic attack)



Figură 5.50 – Reprezentarea vizuală a trasărilor de linii (1)

Atacului greu (Heavy attack)



Figură 5.51 – Reprezentarea vizuală a trasărilor de linii (2)

Pe lângă aceste două tipuri de atac, sunt incluse și celelalte atacuri din cadrul **Atacurilor Contextuale**. **Atenție!!!** Procesul de notificare al momentelor de atac se marchează în **animația de montaj** al fiecărui atac separat.

5.2.4. Daune Jucător-Oponent

Acum că avem o imagine mai clară asupra modului de trasarea a liniilor de atac. Funcționalitatea Daune jucător – Oponent AI va fi una foarte simplă, deoarece trasarea liniilor de atac vor face posibilă condiția de coliziune, iar ulterior daunele aplicate actorilor.

Transmiterea daunelor de către jucător asupra oponenților

Modul prin care oponenții AI vor primi daune se datorează celor două noduri funcție, **applyDamage()** [29] și **eventAnyDamage()** [30], care funcționează în strânsă legătură. [Figură 5.52]



Figură 5.52 – Nodurile de transmitere și primire daune

Funcția applyDamage() este utilizată pentru a trimite o cantitate de daune de la sursa atacator, către destinația referinței actorului atacat.

Funcția eventAnyDamage() primește cantitatea daunelor provocate de sursa atacator, facilitând mai departe logica și direcția în care se îndreaptă evenul.



Figură 5.53 – Transmiterea daunelor de către jucător asupra oponenților (Blueprint)

In continuare, după ce am facut trasările de linii și avem o atingere de coliziune cu un Pawn, prin nodul **Break Hit Result** filtrăm conținutul extragerii de referință a actorului detectat de coliziunea liniilor și aplicarea unei cantități de daune (Base Damage), actorului (Damaged Actor) prin pinii aferenți funcției de **applyDamage()**. [Figură 5.53]

```

În momentul atacului
  IF(Am întâlnit o notificare în animația de montaj == True), THEN
    Executăm primirea de notificare
    IF(jucătorul are implementată interfață == True), THEN
      DO
        | Obținem poziția de start a armei
        | Obținem poziția de sfârșit a armei
        | Trasăm o sferă de coliziune între cele 2 puncte
        | IF(am atins un actor), THEN
          |   applyDamage()
          |   delay(0.2)
        END IF
      END WHILE (notificare activă(animația a fost completată?))
      Executăm sfârșitul de notificare
      IF (jucătorul are implementată interfață == True), THEN
        | Resetăm notificarea
      END IF
    END IF
  END IF

```

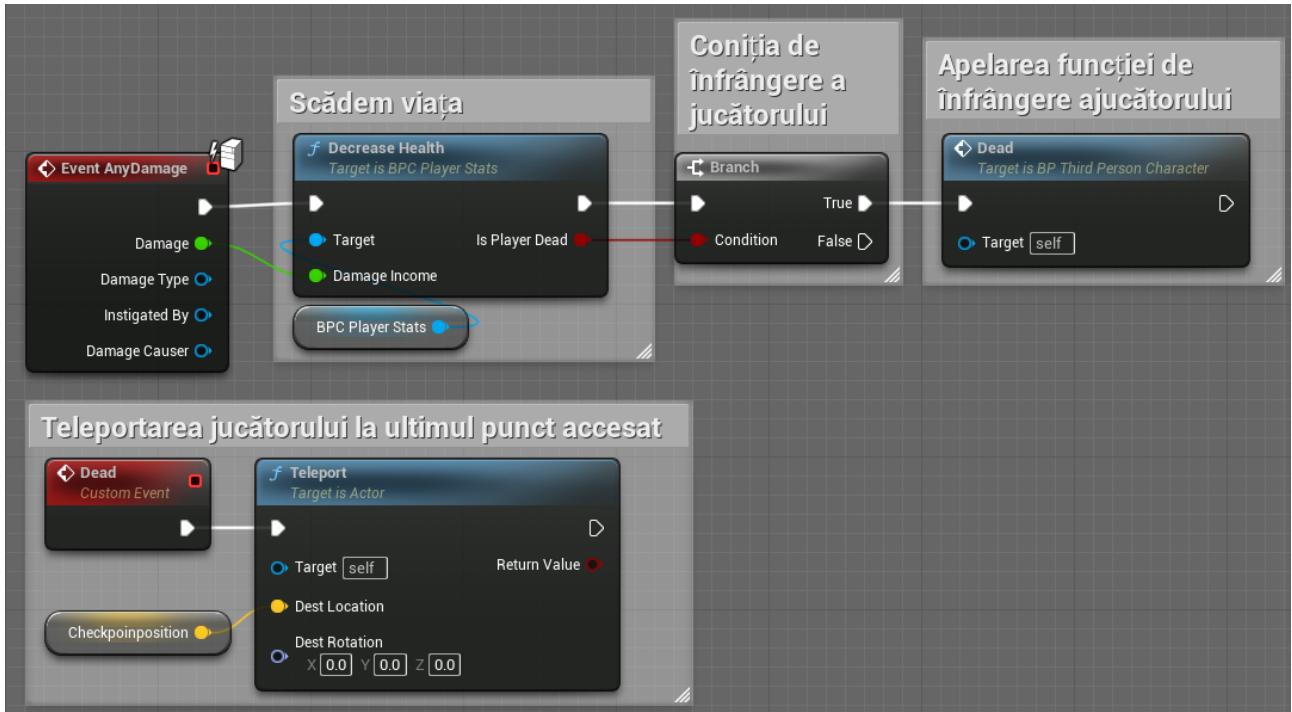
Prin implementarea acestei logici de detectie am realizat funcționalitate de daune. Totuși sistemul nu va funcționa corect, **independent**. Acest aspect de rezolvare al problemei este evidențiat în cadrul **Sistemului de Control AI**.



Figură 5.54 – Reprezentarea vizuală a coliziunilor de trasare linii

Recepția daunelor de către jucător

După cum știm, responsabilul event de aplicare a daunelor unui actor este **eventAnyDamage()**. Iar prin această logică de funcționare vom determina rezultatul ulterior al condiției de pierdere a jucătorului.



Figură 5.55 – Recepționarea daunelor de către jucător (Blueprint)

```

În momentul în care jucătorul primește daune de la un oponent
decreaseHealth()
IF(isPlayerDead == True), THEN
|   Teleportăm jucătorul la poziția de start sau la ultimul punct
|   accesat
END IF

```

decreaseHealth() este o funcție folosită pentru scăderea vieții curente a jucătorului și care returnează condiția booleană **isPlayerDead**. Condiția de adevar v-a reprezenta înfrângerea jucătorului și apelarea funcției **dead()**, rezultând teleportarea acestuia la coordonatele de început a startului de joc sau ultima accesare a unui punct de access „Checkpoint”.

5.3. Sub-sistemul de attribute

Sub-sistemul este o componentă esențială a unui joc, deoarece aceasta reprezintă caracteristicile unui personaj cum ar fi: viață, rezistență fizică, dar și alte attribute în funcție de direcția jocului propriu-zis, pentru a influența modul prin care un actor interacționează cu mediul.

Sub-sistemul de attribute este creat astfel încât să fie o **componentă de actor**. Cu alte cuvinte reutilizabilă și pentru alți actori ca de exemplu: actorului jucător și actorii oponenți.

După cum am menționat, un sub-sistem de attribute poate fi destul de complex facilitând o multitudine de funcționalități pentru fiecare tip de atribut. Însă am reușit să surprindem doar attributele esențiale și să le folosim în funcție de contextul fiecărui, aceastea fiind următoarele:

- Viață (Health)
- Rezistență fizică (Stamina)
- Experiență (Experience/XP)
- Puterea fizică (Strength)

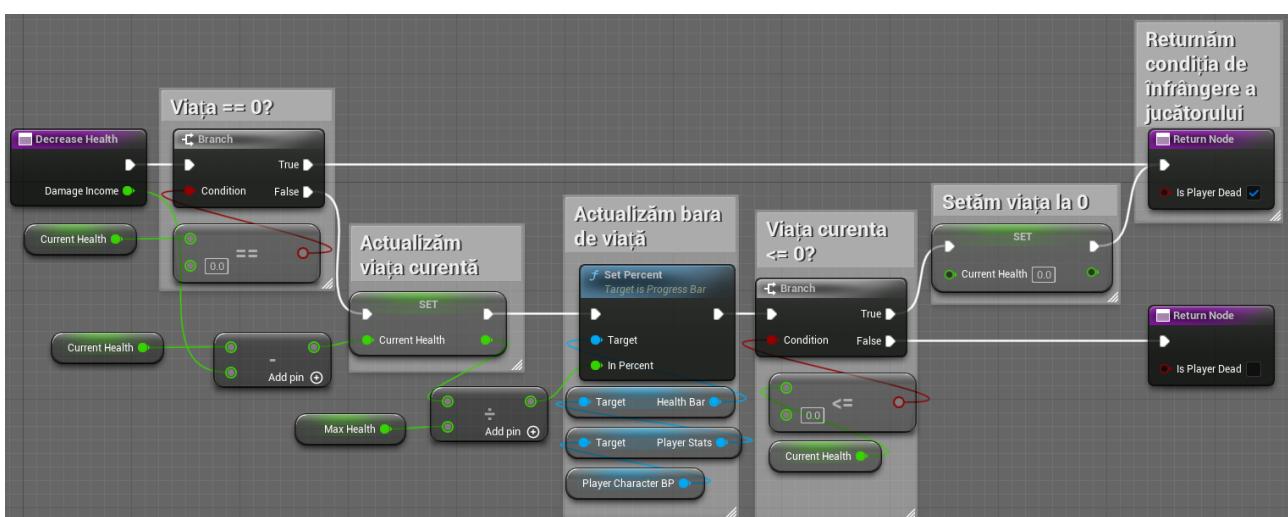
Fiecare atribut având funcții specifice pentru diferite cazuri de utilizare.

5.3.1. Viață, rezistență fizică și experiență jucătorului

Atributul de viață al jucătorului

Atributul de viață al jucătorului este alcătuit din trei funcții esențiale pentru funcționarea corectă a acestuia: **decreaseHealth()**, **increaseHealth()** și **increaseMaxHealth()**.

Funcția **decreaseHealth()** este folosită pentru reducerea unui procent din viață curentă a jucătorului, în general fiind utilizată pentru daunele provocate de către un oponent și returnarea unui răspuns de tip boolean dacă jucătorul a fost învins sau nu în momentul în care viața ajunge la valoarea zero. [Figură 5.56]



Figură 5.56 – Funcția de reducere a vieții (Blueprint)

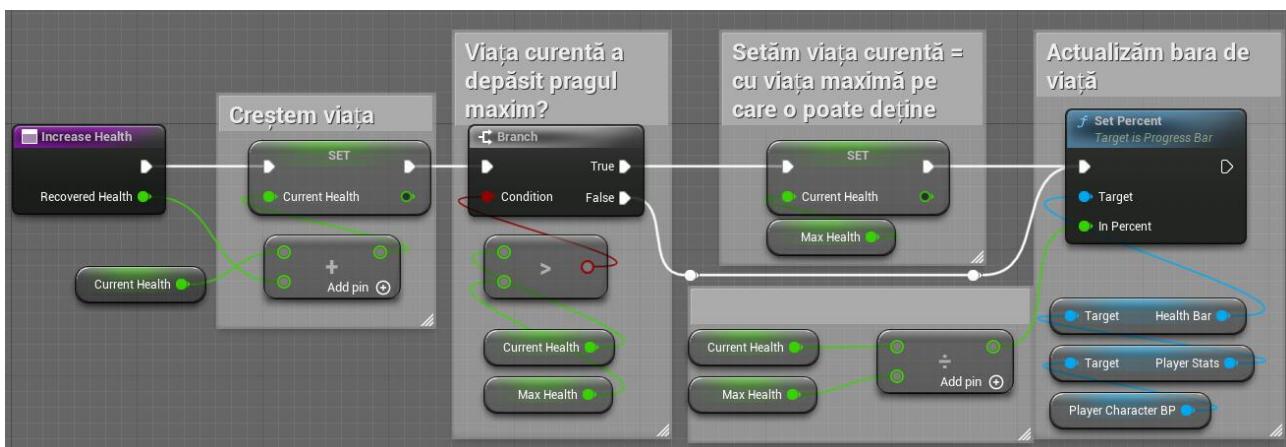
```

În momentul apelării funcției decreaseHealth()
    IF(currentHealth == 0), THEN
        | isPlayerDead = True //jucătorul a fost învins
    ELSE
        | currentHealth = currentHealth - damageIncome
        | actualizăm bara de viață, currentHealth % maxHealth
        | IF(currentHealth <= 0), THEN
        |     | currentHealth = 0
        |     | isPlayerDead = True
        | ELSE
        |     | isPlayerDead = False //jucătorul nu a fost încă invins
    END IF
END IF

```

- **currentHealth** – variabilă de tip float care stochează viața curentă a jucătorului.
- **damageIncome** – valoare de tip float care indică cantitatea de daune primite, pentru a scădea procentul de viață al jucătorului.
- **isPlayerDead** – variabilă de tip boolean utilizată pentru a indica momentul de învingere a jucătorului.
- **maxHealth** – variabilă de tip float care tine evidența vieții maxime pe care o poate deține jucătorul.

Funcția **increaseHealth()** este utilizată pentru restaurarea vieții jucătorului prin adăugarea unui procent la viața curentă. [Figură 5.57]



Figură 5.57 – Funcția de creștere a vieții (Blueprint)

```

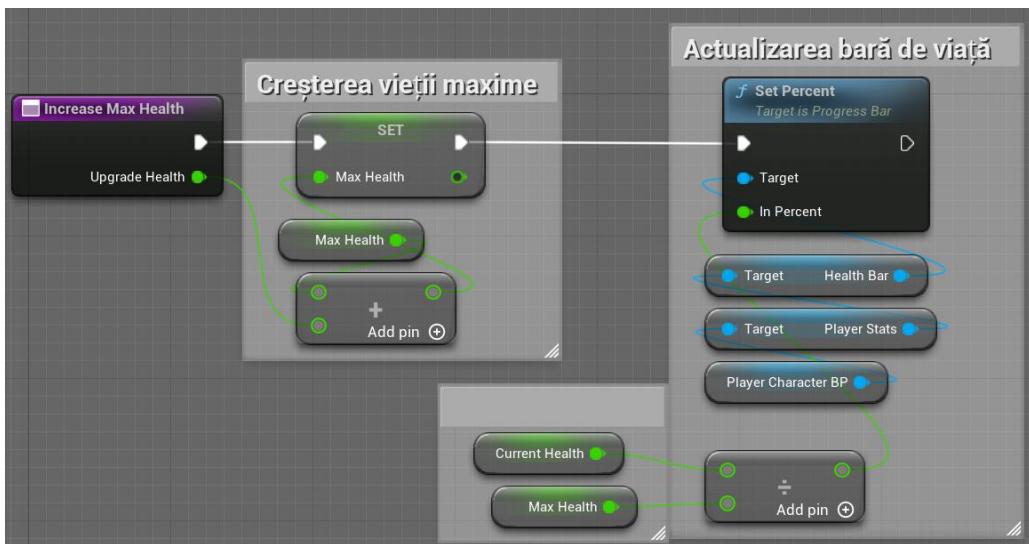
În momentul apelării funcției increaseHealth()
    currentHealth = currentHealth + recoveredHealth
    IF(currentHealth > maxHealth), THEN
        | currentHealth = maxHealth
    END IF
    Actualizăm bara de viață

```

- **recoveredHealth**: valoare de tip float responsabilă cu restaurea procentului de viață.

Funcția **increaseMaxHealth()** este utilizată pentru modificarea vieții maxime a jucătorului prin creșterea valorii maxime de viață pe care o poate avea acesta. [Figură 5.58]

Figură 5.58 – Funcția de creștere a vieții maxime (Blueprint)



**În momentul apelării funcției
increaseMaxHealth()**

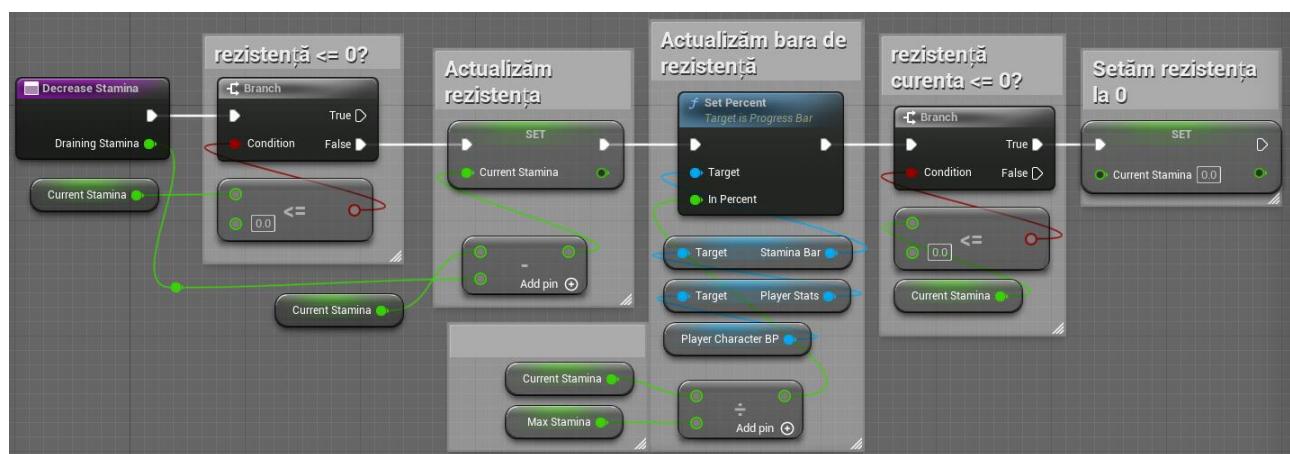
```
maxHealth = maxHealth + upgradeHealth
Actualizăm bara de viață
```

- **upgradeHealth**: valoare de tip float responsabilă de creșterea valorii de viață maximă pe care o poate avea jucătorul.

Atributul de rezistență fizică al jucătorului

Atributul de rezistență fizică al jucătorului este alcătuit din trei funcții esențiale pentru funcționarea corectă a acestuia: **decreaseStamina()**, **increaseStamina()** și **increaseMaxStamina()**.

Funcția **decreaseStamina()** este folosită pentru reducerea unui procent din rezistență fizică curentă a jucătorului, în general fiind utilizată în cadrul atacurilor, sau acțiunilor de tip săritură sau evitare până la pragul de zero. [Figură 5.59]



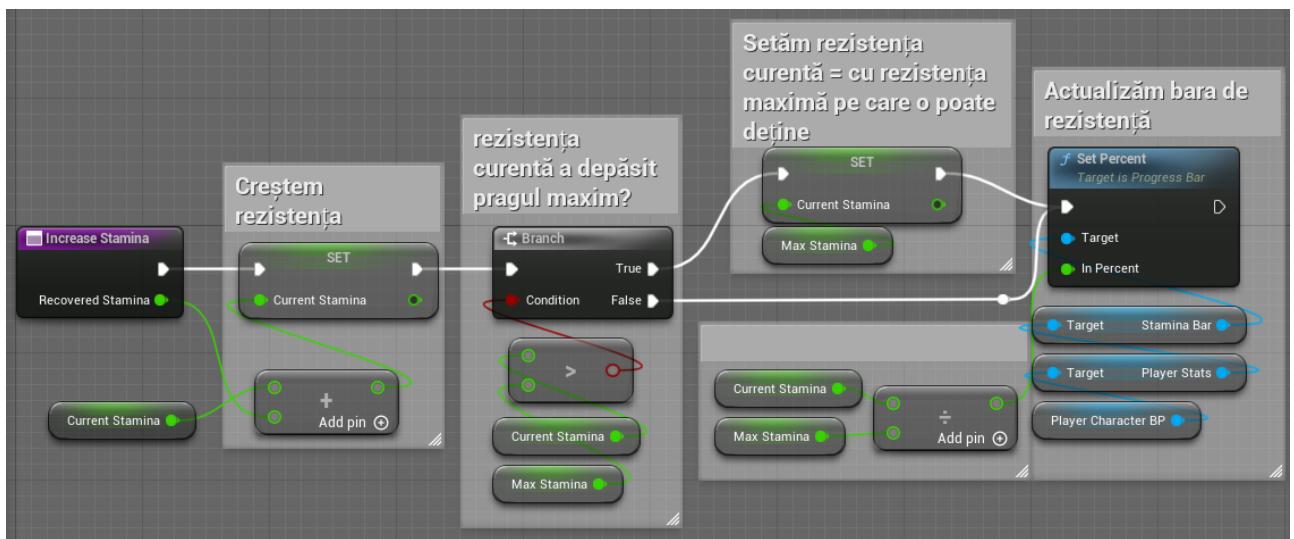
Figură 5.59 – Funcția de reducere a rezistenței fizice (Blueprint)

- **currentStamina** – variabilă de tip float care stocăreză rezistență curentă a jucătorului.
- **drainingStamina** – valoare de tip float care indică cantitatea de rezistență scăzută.

- **maxStamina** – variabilă de tip float care tine evidența rezistenței fizice maxime pe care o poate deține jucătorul.

```
În momentul apelării funcției decreaseStamina()
  IF((currentStamina <= 0) == False), THEN
    |   currentStamina = currentStamina - drainingStamina
    |   actualizăm bara de rezistență fizică, currentStamina %
      maxStamina
    |   IF(currentStamina <= 0) == True), THEN
    |     |   currentStamina = 0
    |   END IF
  END IF
```

Funcția **increaseStamina()** este utilizată pentru restaurarea rezistenței fizice a jucătorului prin adăugarea unui procent la viața curentă. [Figură 5.60]

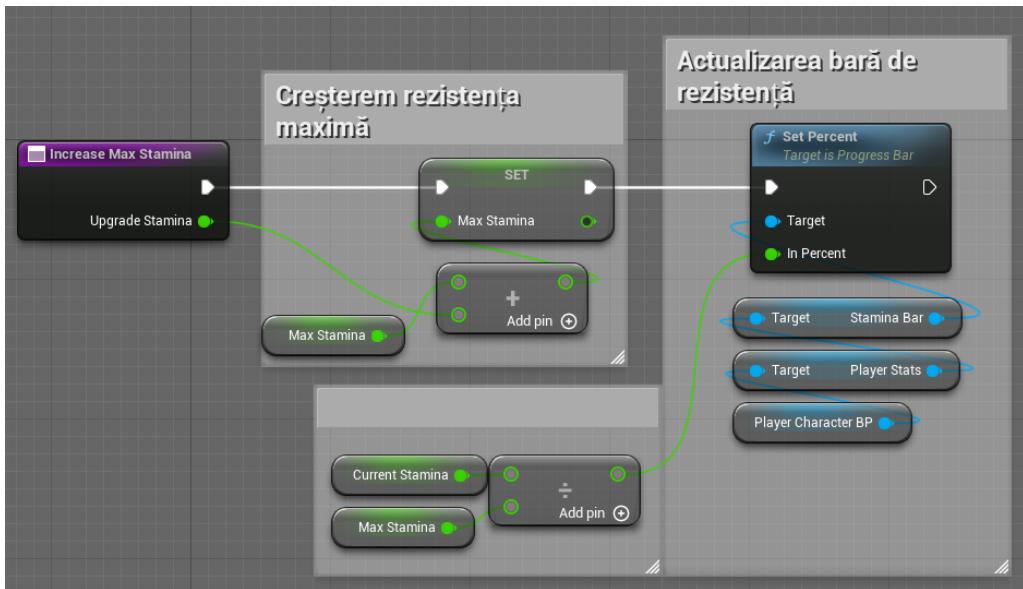


Figură 5.60 – Funcția de creștere a rezistenței fizice (Blueprint)

```
În momentul apelării funcției increaseStamina()
  currentStamina = currentStamina + recoveredStamina
  IF(currentStamina > maxStamina), THEN
    |   currentStamina = maxStamina
  END IF
  Actualizăm bara de rezistență
```

- **recoverStamina** – valoare de tip float responsabilă cu restaurarea procentului de rezistență.

Funcția **increaseMaxStamina()** este utilizată pentru modificarea rezistenței fizice maxime a jucătorului prin creșterea valorii maxime de rezistență pe care o poate avea acesta.



Figură 5.61 – Funcția de creștere a rezistenței fizice maxime (Blueprint)

**În momentul apelării funcției
increaseMaxStamina ()**

```
maxStamina = maxStamina + upgradeStamina
Actualizăm bara de rezistență
```

- **upgradeStamina**: valoare de tip float responsabilă de creșterea valorii de rezistență maximă pe care o poate avea jucătorul.



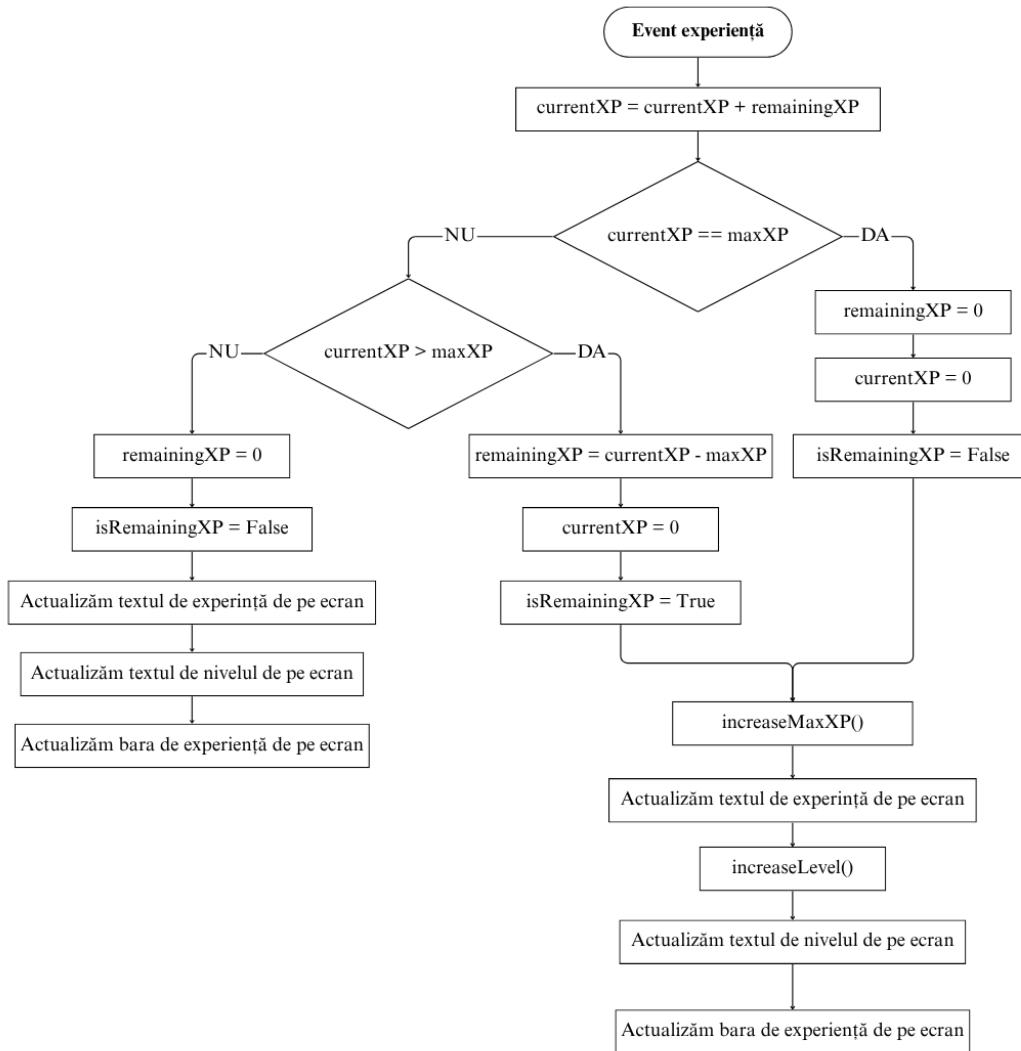
Figură 5.62 – Bara de viață și rezistență fizică

Atributul de experiență al jucătorului

Atributul de experiență este folosit pentru a indica progresul jucătorului, pe baza unor acțiuni desfășurate cum ar fi, înfrângerea de oponenți sau finalizarea de misiuni.

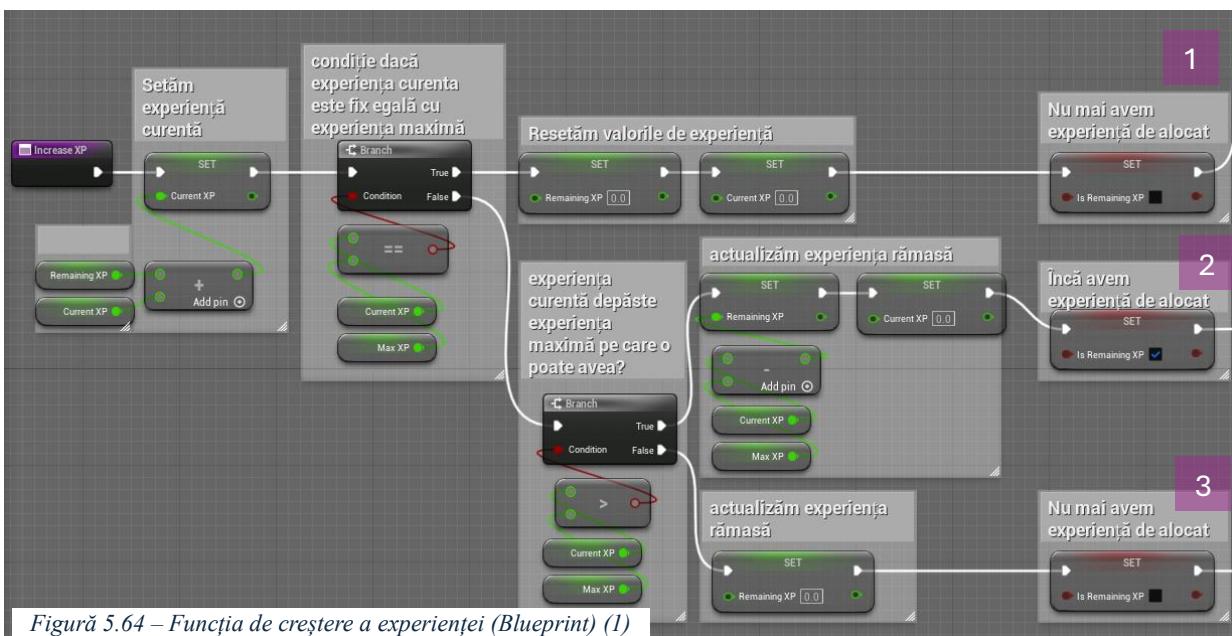
Este alcătuit din trei funcții esențiale pentru funcționarea corectă a acestuia: **increaseXP()**, **increaseMaxXP()** și **increaseLevel()**.

Atributul de experiență poate fi puțin mai complex și diferit față de celelalte menționate mai sus facilitând un altfel de rol. Astfel o să detaliem printr-o organigramă mecanismul de funcționare al acesteia:

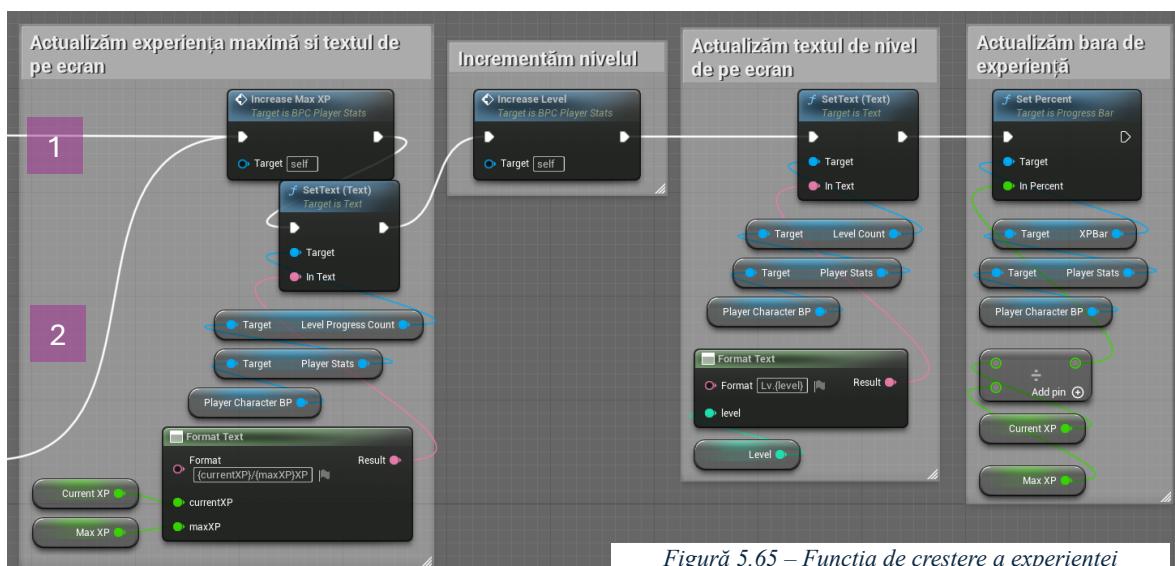


Figură 5.63 – Organigramă generală a atributului de experiență

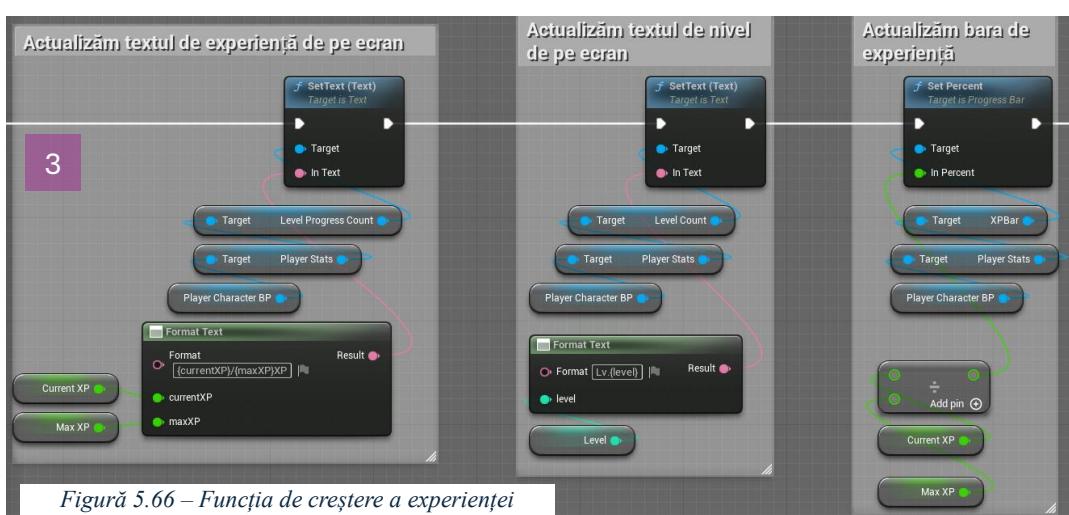
Această organigramă detaliază modul prin care se realizează gestionarea de experiență, creșterea maximă de alocare pentru diversificare și obținerea de recompensă tip puncte în urma îndeplinirii condiției de creștere în nivel.



Figură 5.64 – Funcția de creștere a experienței (Blueprint) (1)



Figură 5.65 – Funcția de creștere a experienței



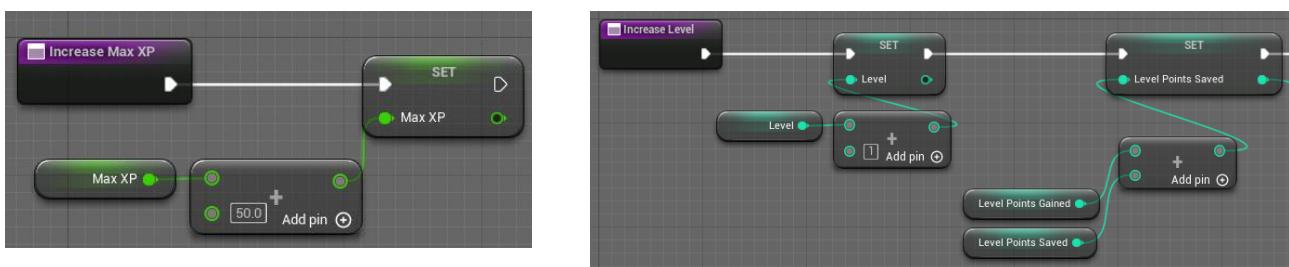
```

În momentul apelării funcției increaseXP()
    currentXP = currentXP + remainingXP
    IF(currentXP == maxXP), THEN
        | remainingXP = 0
        | currentXP = 0
        | isRemainingXP = False
        | increaseMaxXP()
        | Actualizăm textul experiență maximă de pe ecran
        | increaseLevel()
        | Actualizăm textul nivel de pe ecran
        | Actualizăm bara de experiență de pe ecran
    ELSE
        | IF(currentXP > maxXP), THEN
            | remainingXP = currentXP - maxXP
            | currentXP = 0
            | isRemainingXP = True
            | increaseMaxXP()
            | Actualizăm textul experiență maximă de pe ecran
            | increaseLevel()
            | Actualizăm textul nivel de pe ecran
            | Actualizăm bara de experiență de pe ecran
        ELSE
            | RemainingXP = 0
            | isRemainingXP = False
            | Actualizăm textul experiență maximă de pe ecran
            | Actualizăm textul nivel de pe ecran
            | Actualizăm bara de experiență de pe ecran
        END IF
    END IF

```

CurrentXP și remainingXP sunt două variabile de tip float care lucrează în strânsă legătură, folosind un mecanism de interschimbare cu două funcționalități:

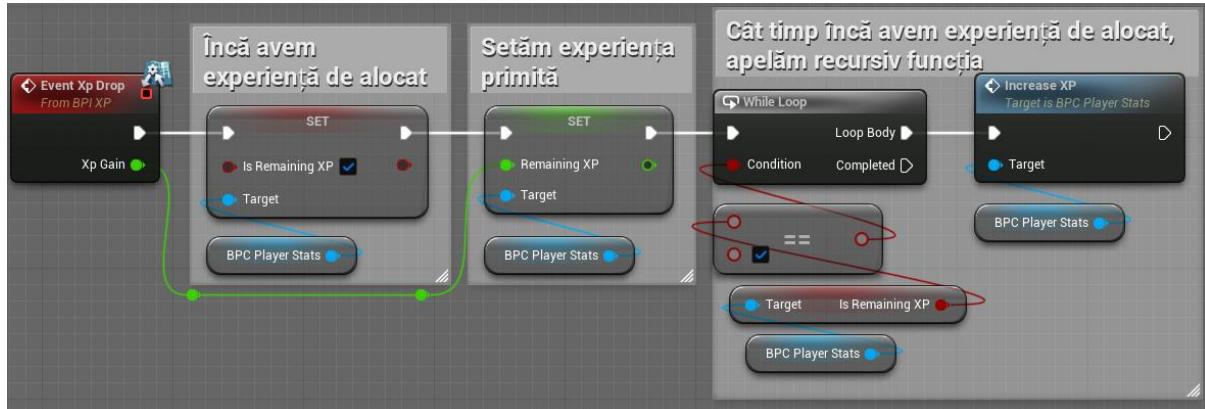
- **remainingXP**: determină cantitatea de experiență nou alocată variabilei currentXP
- **currentXP**: variabilă de tip float care indică experiența curentă.
- În al doilea caz, remainingXP va reprezenta **cantitatea de experiență RĂMASĂ pentru a seta currentXP**.
- Acest mod de reprezentare este util în momentul în care primim o cantitate de experiență mai mare decât poate gestiona sistemul. **Pentru a nu permite pierderea de experiență**.
- **maxXP**: variabilă de tip float care indică cantitatea de experiență maximă pe care o poate atinge.
- **isRemainingXP**: variabilă de tip boolean pentru a valida dacă a mai rămas experiență de alocat sau nu.
- **Funcția increaseMaxXP()**: utilizată pentru incrementarea numărului maxim curent de experiență pe care îl poate atinge jucătorul cu 50.



Figură 5.67 – Funcția de creștere a experienței maxime (Blueprint)

- **Functia increaseLevel()**: utilizată pentru incrementarea de nivel cu unu a jucătorului și primirea de puncte recompensă.
- **level**: variabilă de tip integer, folosită pentru a indica nivelul jucătorului.
- **levelPointsGained**: variabilă de tip integer utilizată pentru atribuirea punctelor de recompenșă ale jucătorului.
- **levelPointsSaved**: variabila de tip integer utilizată pentru a salva punctele de recompensă

Mecanismul prin care jucătorul primește un eveniment de experiență de la o sursă, va fi apelat în mod repetat (recursiv) funcția **increaseXP()** până când cantitatea de experiență nu va mai fi posibil alocată, datorită pragului de insuficiență. [Figură 5.68]

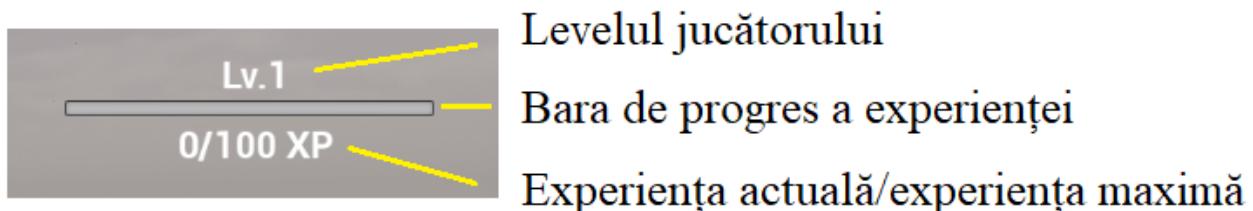


Figură 5.68 – Funcția de primire experiență (Blueprint)

```

În momentul apelării funcției xpDrop()
    isRemainingXP = True
    remainingXP = xpGain
    WHILE(isRemainingXP == True) DO
        |   increaseXP()
    END WHILE
  
```

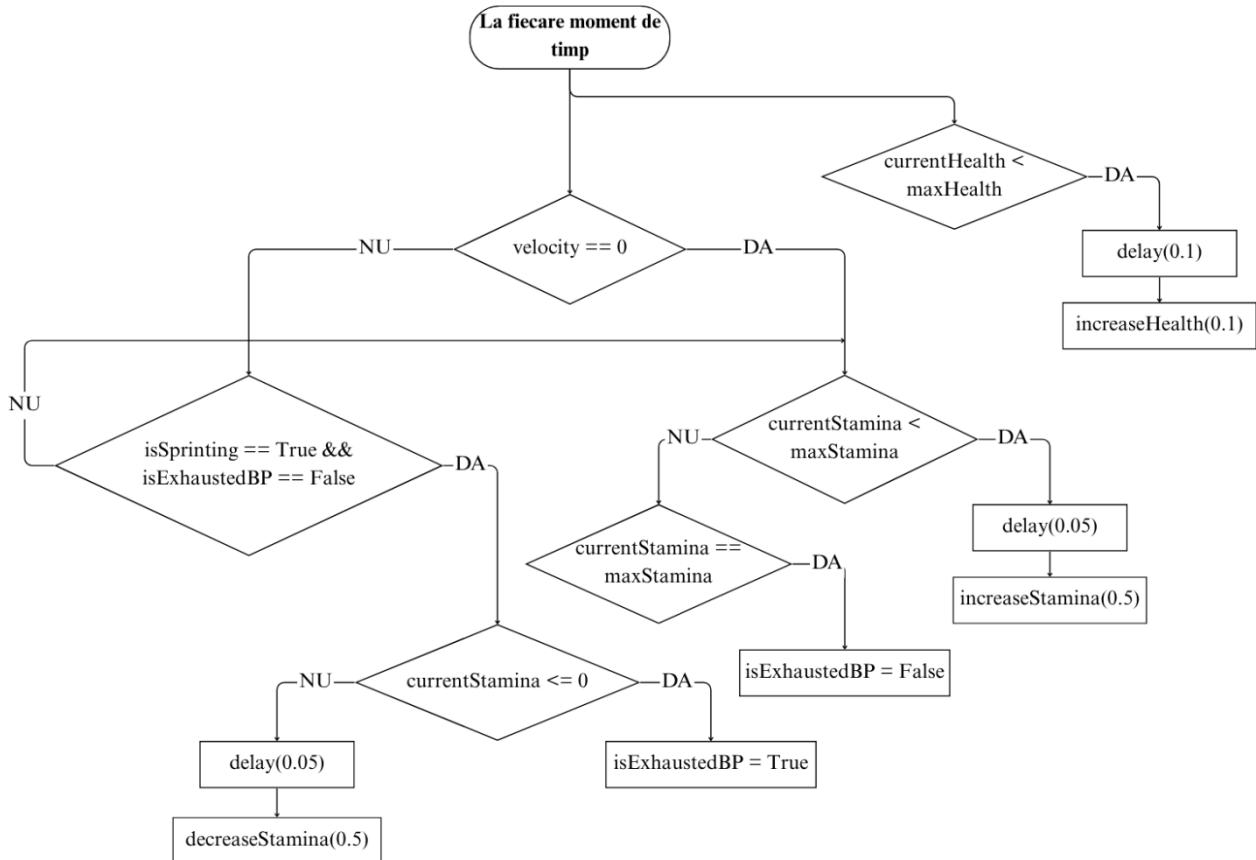
- **xpGain**: variabilă de tip float responsabilă cu returnarea unei cantități de experiență.



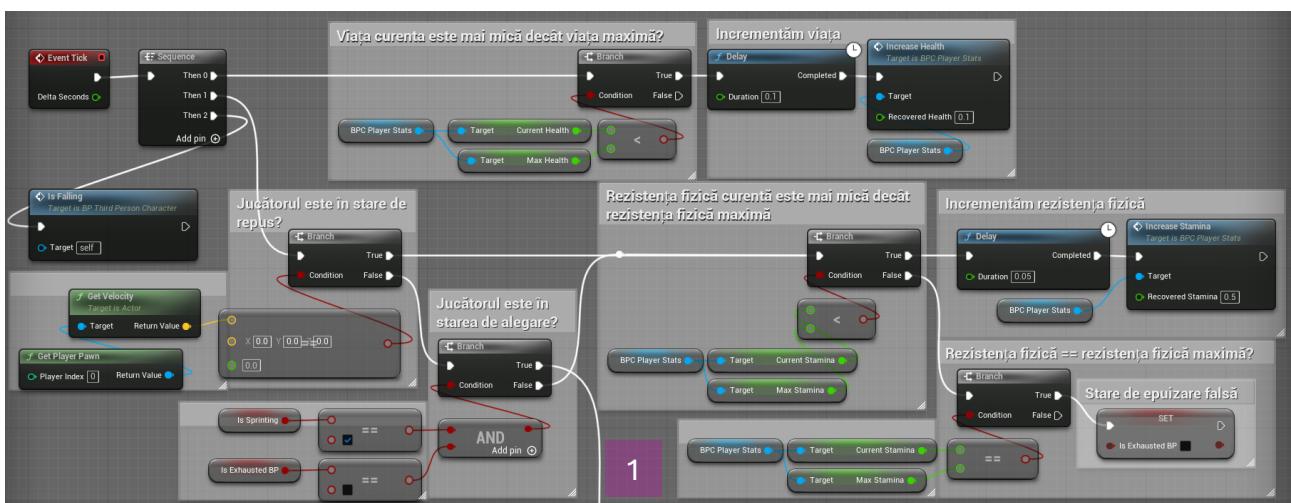
Figură 5.69 – Bara de experiență

5.3.2. Recuperarea de atribute ale jucătorului

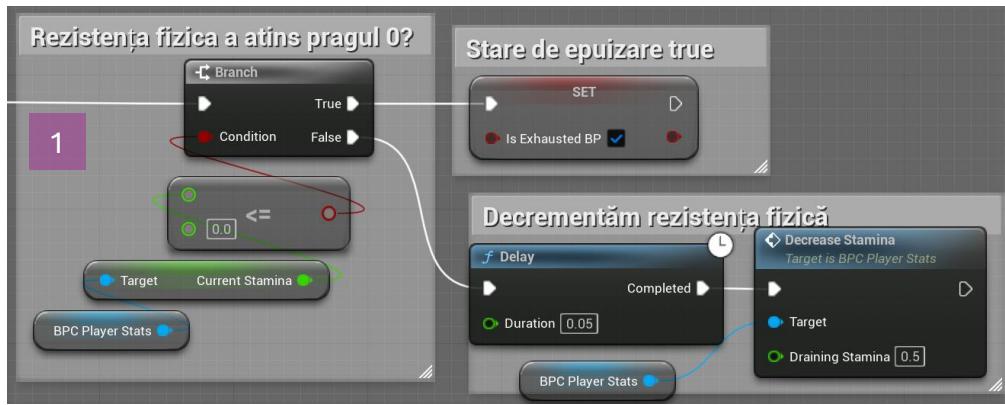
Recuperarea de atrbute ale jucătorului reprezintă procesul prin care valorile statisticilor precum viață sau rezistență sunt restituite parțial sau complet. Este un mecanism esențial în jocuri, folosit atât în limitarea acțiunilor jucătorului cât și în echilibrarea experienței de joc.



Figură 5.70 – Organograma generală a recuperării de atrbute ale jucătorului



Figură 5.71 – Recuperare de attribute ale jucătorului (Blueprint) (1)



Figură 5.72 – Recuperare de atribute ale jucătorului (Blueprint) (2)

```

La fiecare moment de timp
//incrementăm viață
IF(currentHealth < maxHealth), THEN
| Delay(0.1)
| increaseHealth(0.1)
END IF
//dacă suntem în starea de repaus
IF(velocity == 0), THEN
| IF(currentStamina < maxStamina), THEN
| | delay(0.05)
| | increaseStamina(0.5) //incrementăm rezistență fizică
| ELSE
| | IF(currentStamina == maxStamina), THEN
| | | isExhaustedBP = False
| | END IF
| END IF
ELSE
| IF(((isSprinting == True) AND (isExhausted == False)) == True),
THEN
| | IF(currentStamina <= 0), THEN
| | | isExhaustedBP = True //personajul este extenuat
| | ELSE
| | | delay(0.05)
| | | decreaseStamina(0.5) //decrementăm rezistență
| | END IF
| ELSE
| | IF(currentStamina < maxStamina), THEN
| | | delay(0.05)
| | | increaseStamina(0.5) //incrementăm rezistență
| | ELSE
| | | IF(currentStamina == maxStamina), THEN
| | | | isExhaustedBP = False
| | | END IF
| | END IF
END IF
END IF

```

5.4. Sub-sisteme suport

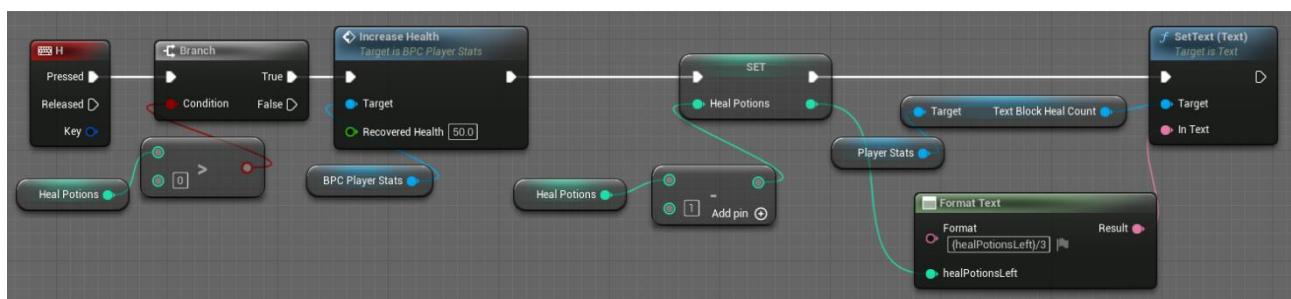
5.4.1. Consumabile

Consumabilele sunt obiecte utilizate de către jucător, care oferă efecte temporare sau de lungă durată. Acestea joacă un rol important în gestionarea de resurse și supraviețuirea pe lungă durată.

Poțiunea de viață



Figură 5.73 – Poțiunea de viață



Figură 5.74 – Poțiunea de viață (Blueprint)

```
La apăsarea tastei „H”
    IF(healPotion > 0), THEN
        |     increaseHealth(50)
        |     healPotion = healPotion - 1
        |     Actualizăm textul de pe ecran care
            contorizează poțiunile
    END IF
```

Jucătorul își va putea reface viață prin folosirea unei poțiuni și decrementare cu unu din numărul total pe care le poate deține.

- **healPotion:** variabilă de tip integer responsabilă în contorizarea numărului de poțiuni rămase.

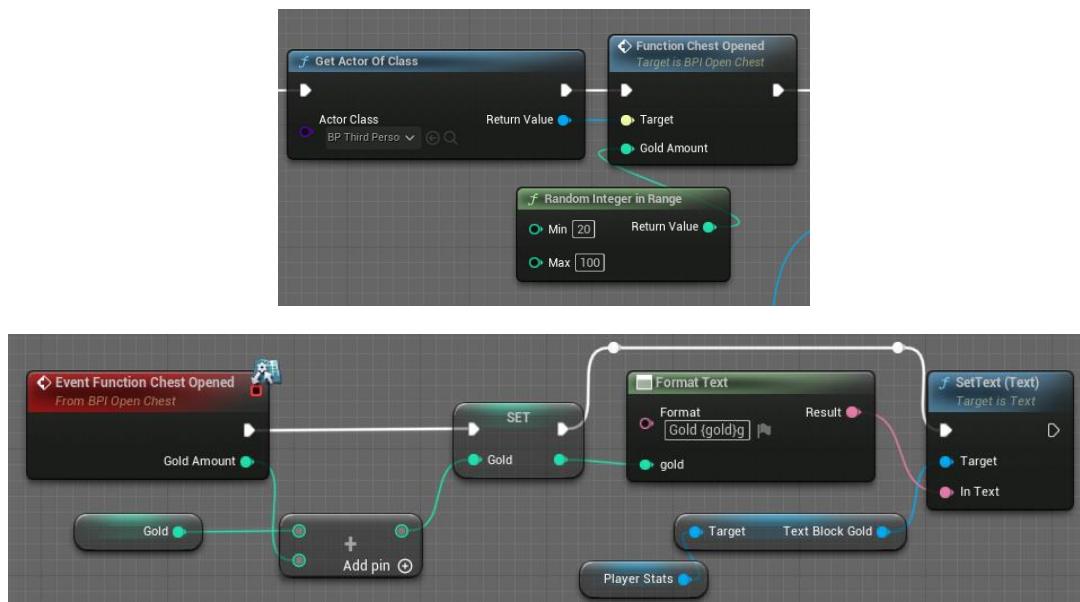
5.4.2. Recompense

Recompensele sunt introduse cu scopul de a încuraja progresul jucătorilor și de a le oferi satisfacție în urma îndelphinirii unor acțiuni, condiții sau sarcini îndeplinite, etc. Cum ar fi:

- Recompense din cufere
- Recompense din misiuni
- Recompense din experiență

Recompense din cufere

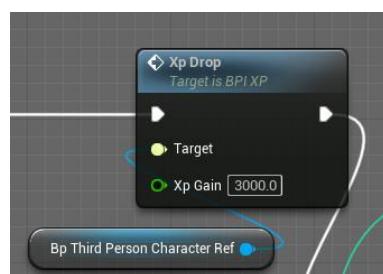
Recompensele din cufere sunt dobândite în momentul în care jucătorul interacționează cu un cufăr, pe care acesta îl deschide primind o **recompensă aleatoare între 20 și 100 gold (aur)**. Aurul reprezentând moneda de schimb, utilizată pentru achiziții din magazinele personajelor NPC (Non-Player Character).



Figură 5.75 – Funcția de recompense din cufere (Blueprint)

Recompense din misiuni

Recompensele din misiuni sunt dobândite prin completarea unor sarcini, cum ar fi: „Vorbește cu ...”, „Colectază ...” sau „Învinge ...”. Sub formă de experiență prin apelarea funcției **xpDrop()** la sfârșitul completării.



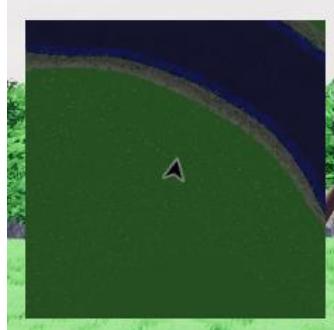
Figură 5.76 – Funcția de repompense (Blueprint)

Recompense din experiență

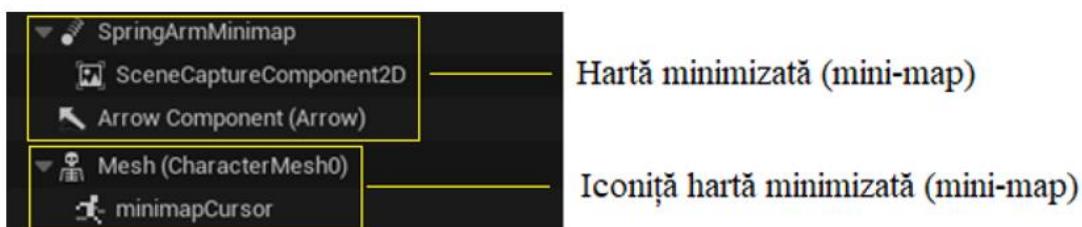
Recompensele din experiență sunt dobândite prin atingerea capacitatii maxime de experiență în funcție de nivel-ul propriu-zis. Dacă această condiție a fost îndeplinită, atunci jucătorul va primi și recompensă, **puncte de nivel**. Aceste puncte pot fi folosite cu scopul de a îmbunătății atributile fizice ale jucătorului, precum: viață, rezistență și putere.

5.4.3. Harta minimizată a jucătorului (Mini-Map)

Harta minimizată, reprezintă un element esențial al interfeței jucătorului facilitând mijlocul prin care acesta poate naviga prin nivel având posibilitatea de a observa poziția curentă, oponenții, terenul și obstacolele mediului virtual. [Figură 5.77]



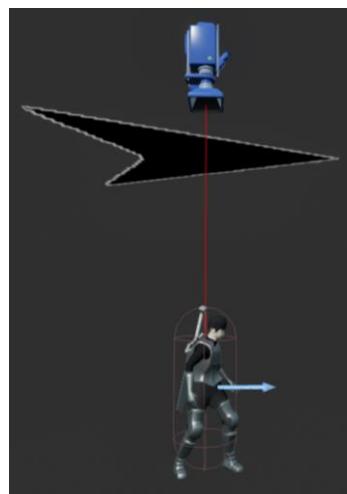
Figură 5.77 – Reprezentare vizuală mini-hartă



Figură 5.78 – Componențele de alcătuire ale mini-hărții

Componențele hărții minizmate

SceneCaptureComponent2D, este un alt tip de cameră utilizat pentru vizualizarea în plan 2D a mediului virtual de joc. [Figură 5.79]



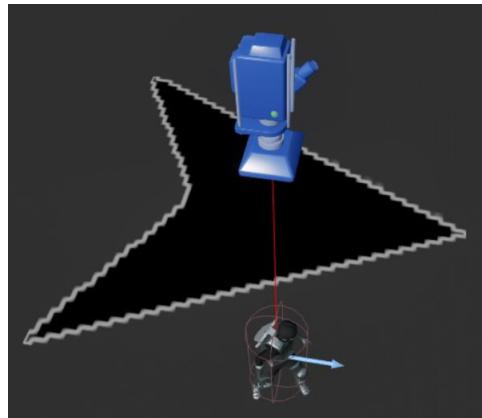
Figură 5.79 – Camera mini-hartă

În proprietățile acestuia, secțiunea de **Texture Target**, creăm o textură de rendering care va servi la captura și actualizarea hărții la fiecare moment de timp și afișarea pe ecran prin intermediul interfeței UI a jucătorului.

În secțiunea pe **Projection**, **Projection Type: Orthographic**, iar **Ortho Width: 4000**.

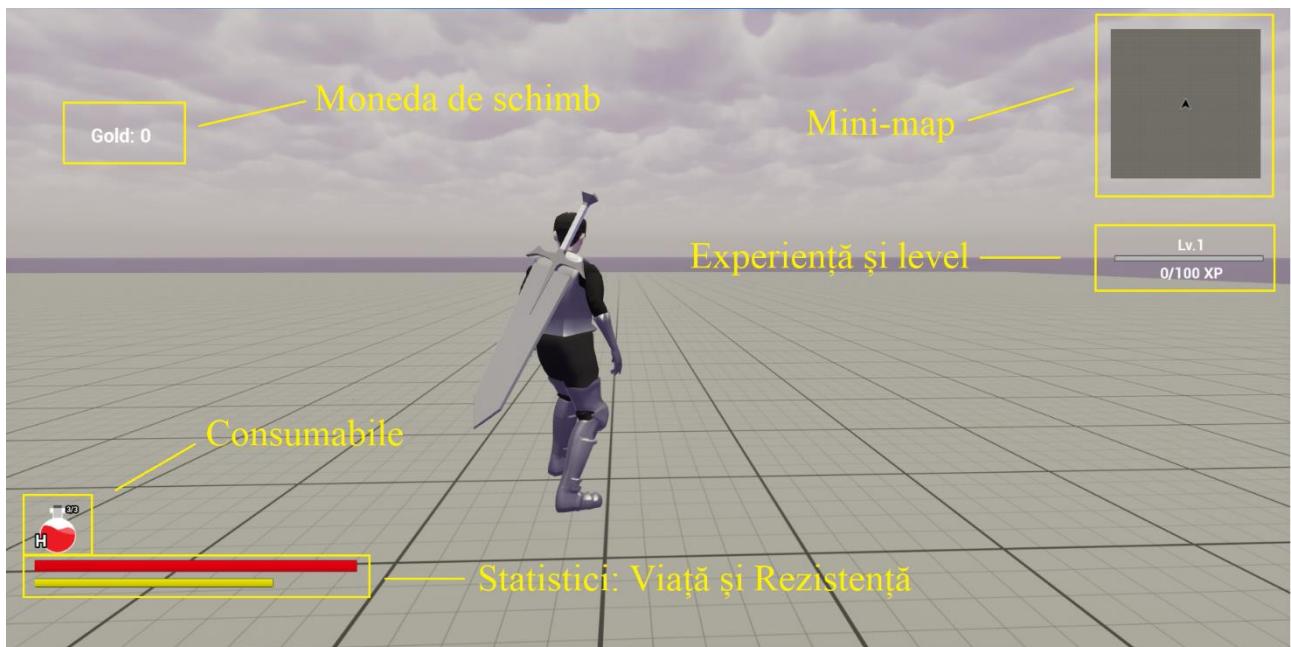
- **Ortho Width:** determină distanța de randare dintre cameră și jucătorul.

MinimapCursor este o imagine de tip **Sprite** poziționat și fixat deasupra jucătorului care va indica direcția în care se deplasează jucătorul. [Figură 5.80]



Figură 5.80 – Cursorul camerei de mini-hartă

Interfața jucătorului.



Figură 5.81 – Interfața jucătorului

Prin elementele enumerate în figura această vor alcătui interfața jucătorului, pentru moment. Urmând să fie integrate și funcționalitățile de **Contorizare a timpului** curent din joc și **Sistemul de Misiuni (Quest-uri)** pe parcursul documentației.

5.5. Sistemul de alcătuirea al Punctului de Control (Checkpoint)

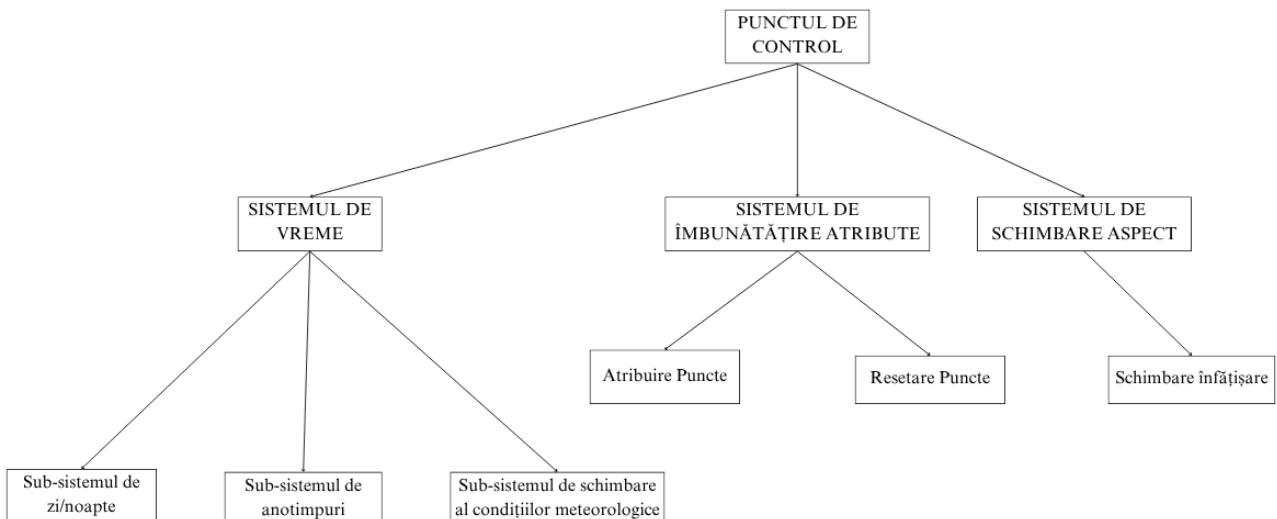
Sistemul constituie una din componentele cele mai complexe și fundamentale ale unui joc. Acest sistem este utilizat frecvent în domeniul dezvoltării jocurilor video oferind multiple funcționalități jucătorului din care să aleagă în funcție de arhitectura de proiectare a acestuia.

Cel mai comun caz este acela de revenire la viață sau teleportare/cale de acces prin care jucătorul poate traversa către o locație. În cadrul acestui sistem reușind să implementăm și următoarele Sisteme și sub-sisteme:

- **Sistemul de vreme**
 - Sub-sistemul de ore – zi/noapte
 - Sub-sistemul de anotimpuri
 - Sub-sistemul de schimb al condițiilor meteorologice
- **Sistemul de îmbunătățiri atribute**
- **Sistemul de schimbare aspect**

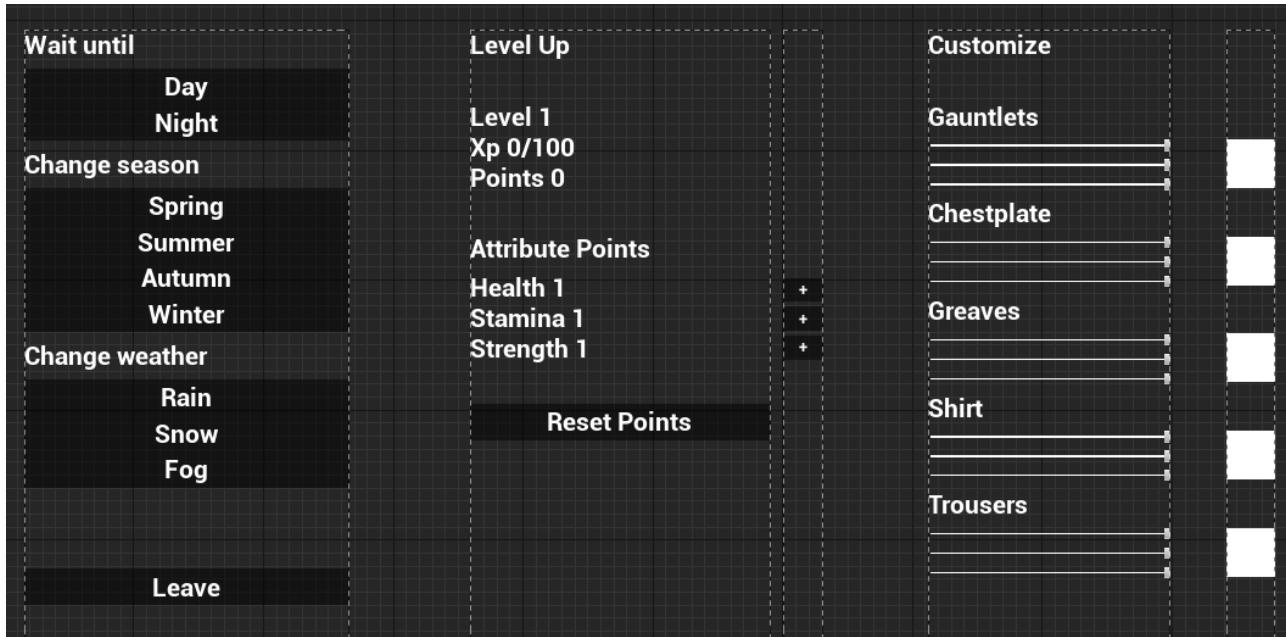
Fiecare Sistem și sub-sistem va fi detaliat în parte pe parcurs.

ORGANIGRAMA GENERALĂ A PUNCTULUI DE CONTROL



Figură 5.82 – Organograma generală a punctului de control

Interfață UI a Punctului de Control



Figură 5.83 – Interfața Widget UI a punctului de control

Interfață UI a punctului de control este alcătuită din mai multe componente cum ar fi: butoane de acțiune, slidere, imagini și text. Fiecare cu un rol specific acestuia.

Interfața pune la dispoziție trei meniuri pe care jucătorul le poate opera și anume: **Sistemul schimbului de vreme**, **Îmbunătățire atributelor** și **Schimbul de înfățișare al personajului**.

- **Sistemul de vreme** înglobează mai multe acțiuni în funcție de preferințele de joc al fiecărui utilizator, prin gestionarea de ore din zi sau noapte, schimburi ale anotimpurilor: primăvară, vară, toamnă, iarnă, dar și a fenomenelor meteorologice cum ar fi: ploaie, ninsoare sau ceată.
- **Sistemul de îmbunătățire a atributelor** permite jucătorului evoluarea personajului în funcție de preferințele și stilul de joc pe care acesta dorește să îl abordeze. Atributele personajului pot fi îmbunătățite prin distribuirea **Punctelor de nivel** la apăsarea butonelor aferente din dreptul fiecăruia. De asemenea **resetarea punctelor** pentru a permite re-specializarea personajului.
- **Sistemul schibului de înfățișare** oferă jucătorului un mod prin care poate modifica aspectul vizual al personajului în funcție de preferința acestuia. Fiecare slider este o componentă RGB pentru a seta o culoare.

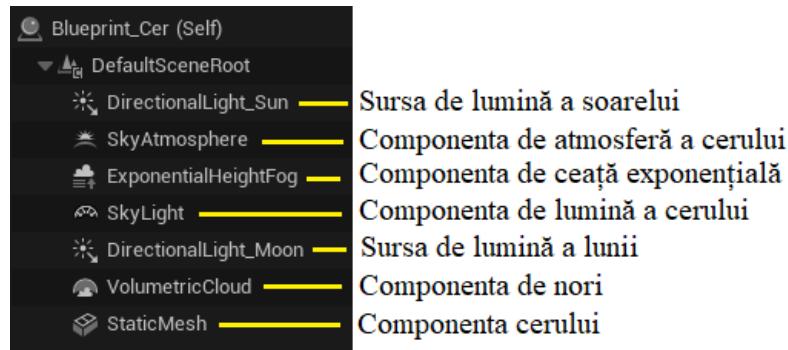
5.5.1. Sistemul de vreme

Sistemul de vreme prin arhitectura acestuia, permite simularea dinamică a mediul înconjurător inclusiv cicluri de zi/noapte, schimbări ale sezoanelor și a condițiilor meteorologice variate oferind un plus de realism în cadrul mediului virtual de joc. Aceasta integrează mai multe sub-sisteme puse la un loc, fiecare responsabil cu moduri de funcționare diferit dar care pot funcționa în paralel sau condiționate.

Sub-sistemul de Ore - Zi/Noapte

Sub-sistemul de ore – zi/noapte, simulează trecerea timpului atât vizual cât și funcțional oferind jucătorului o senzație de lume vie.

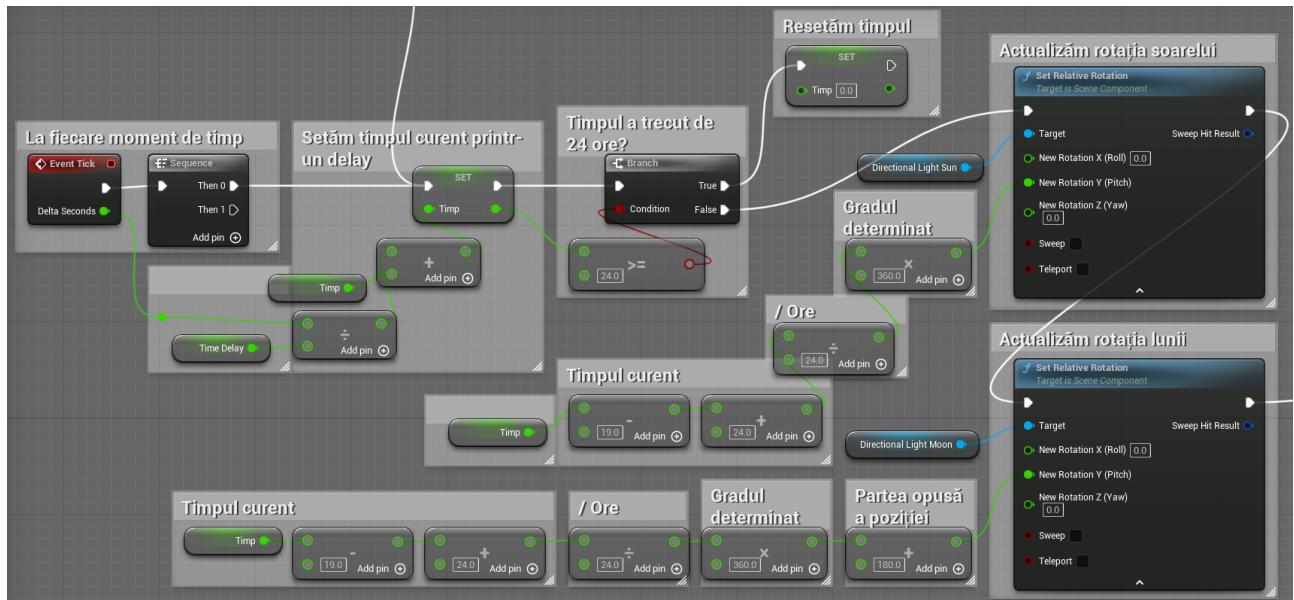
Componentele actorului:



Figură 5.84 – Componente de alcătuire ale actorului cer

- SkyAtmosphere** – creează o atmosferă realistă pentru cerul jocului, efecte precum difuzia luminii, rasărituri și apusuri realiste.
- ExponentialHeightFog** – creează un efect de ceată (fog) care variază în funcție de înălțime, adăugând profunzime și atmosferă scenei.
- SkyLight** – adăugă lumina ambientală globală a scenei de joc.
- VolumetricClouds** – componentă care reprezintă norii de pe cer.
- StaticMesh** – este o componentă de tip sferă care simulează globul care înconjoară suprafața cerului.
- DirectionalLight_Sun și DirectionalLight_Moon** – folosite pentru reprezentarea a celor două surse de lumină soare, respectiv lună.

Blueprint-ul de rotație a celor două surse de lumină [Figură 5.85]



Figură 5.85 – Rotația surselor de lumină (Blueprint)

```

La fiecare moment de timp
    timp = (deltaSeconds / timeDelay) + timp
    IF(timp >= 24), THEN
        |      timp = 0 //pentru resetare
    ELSE
        |      Actualizăm rotația soarelui(((timp - 19) + 24) / 24) * 360)
        |      Actualizăm rotația lunii(((timp - 19) + 24) / 24) * 360) + 180
    END IF

```

- **deltaSeconds** – variabilă de tip float care reprezintă timpul scurs între două frame-uri (cadre).
- **timeDelay** – variabilă de tip float responsabilă cu întarzirea de timp.
- **Timp** – variabilă de tip float responsabilă de contorizarea trecerii a 24 de ore din joc.

Tranzițiile de Zi/Noapte

Tranziția celor două surse de lumină, zi/noapte este controlată prin formula matematică:

$$\text{Rotatie} = (((\text{timp} - 19) + 24) / 24) * 360)$$

Care transformă timpul curent într-un unghi de rotație pe axa Y folosit în cadrul sursei de lumină a soarelui pentru realizarea de tranzitii între orele 07:00 (dimineața) și 19:00 (seara).

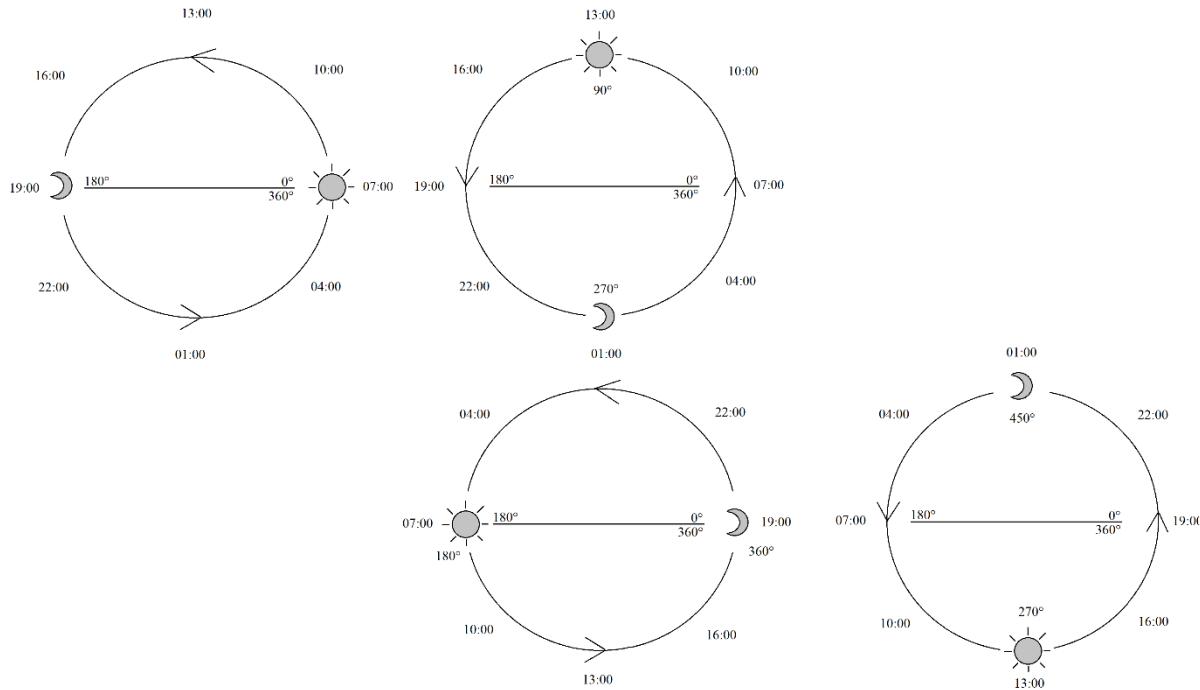
Atenție! Folosim + 180 (grade) pentru rotația de lună deoarece cele două surse de lumină trebuie să fie la distanțe opuse una față de celalaltă.

Temp	Rotatie soare	Rotatie lună
0	$((0-19)+24)/24)*360 = (5/24)*360 = 75^\circ$	$75+180 = 255^\circ$
1	$((1-19)+24)/24)*360 = (6/24)*360 = 90^\circ$	$90+180 = 270^\circ$
2	$((2-19)+24)/24)*360 = (7/24)*360 = 105^\circ$	$105+180 = 285^\circ$
3	$((3-19)+24)/24)*360 = (8/24)*360 = 120^\circ$	$120+180 = 300^\circ$
4	$((4-19)+24)/24)*360 = (9/24)*360 = 135^\circ$	$135+180 = 315^\circ$
5	$((5-19)+24)/24)*360 = (10/24)*360 = 150^\circ$	$150+180 = 330^\circ$
6	$((6-19)+24)/24)*360 = (11/24)*360 = 165^\circ$	$165+180 = 345^\circ$
7	$((7-19)+24)/24)*360 = (12/24)*360 = 180^\circ$	$180+180 = 360^\circ (= 0^\circ)$
8	$((8-19)+24)/24)*360 = (13/24)*360 = 195^\circ$	$195+180 = 375^\circ (= 15^\circ)$
9	$((9-19)+24)/24)*360 = (14/24)*360 = 210^\circ$	$210+180 = 390^\circ (= 30^\circ)$
10	$((10-19)+24)/24)*360 = (15/24)*360 = 225^\circ$	$225+180 = 405^\circ (= 45^\circ)$
11	$((11-19)+24)/24)*360 = (16/24)*360 = 240^\circ$	$240+180 = 420^\circ (= 60^\circ)$
12	$((12-19)+24)/24)*360 = (17/24)*360 = 255^\circ$	$255+180 = 435^\circ (= 75^\circ)$
13	$((13-19)+24)/24)*360 = (18/24)*360 = 270^\circ$	$270+180 = 450^\circ (= 90^\circ)$
14	$((14-19)+24)/24)*360 = (19/24)*360 = 285^\circ$	$285+180 = 465^\circ (= 105^\circ)$
15	$((15-19)+24)/24)*360 = (20/24)*360 = 300^\circ$	$300+180 = 480^\circ (= 120^\circ)$
16	$((16-19)+24)/24)*360 = (21/24)*360 = 315^\circ$	$315+180 = 495^\circ (= 135^\circ)$
17	$((17-19)+24)/24)*360 = (22/24)*360 = 330^\circ$	$330+180 = 510^\circ (= 150^\circ)$
18	$((18-19)+24)/24)*360 = (23/24)*360 = 345^\circ$	$345+180 = 525^\circ (= 165^\circ)$
19	$((19-19)+24)/24)*360 = (24/24)*360 = 360^\circ (= 0^\circ)$	$360+180 = 540^\circ (= 180^\circ)$
20	$((20-19)+24)/24)*360 = (25/24)*360 = 15^\circ$	$15+180 = 195^\circ$
21	$((21-19)+24)/24)*360 = (26/24)*360 = 30^\circ$	$30+180 = 210^\circ$
22	$((22-19)+24)/24)*360 = (27/24)*360 = 45^\circ$	$45+180 = 225^\circ$
23	$((23-19)+24)/24)*360 = (28/24)*360 = 60^\circ$	$60+180 = 240^\circ$
24=0	$((24-19)+24)/24)*360 = (29/24)*360 = 75^\circ$	$75+180 = 255^\circ$

Tabel 5.5 – Tabelul rotatiei surselor de lumină

Plus variațiile dintre acestea.

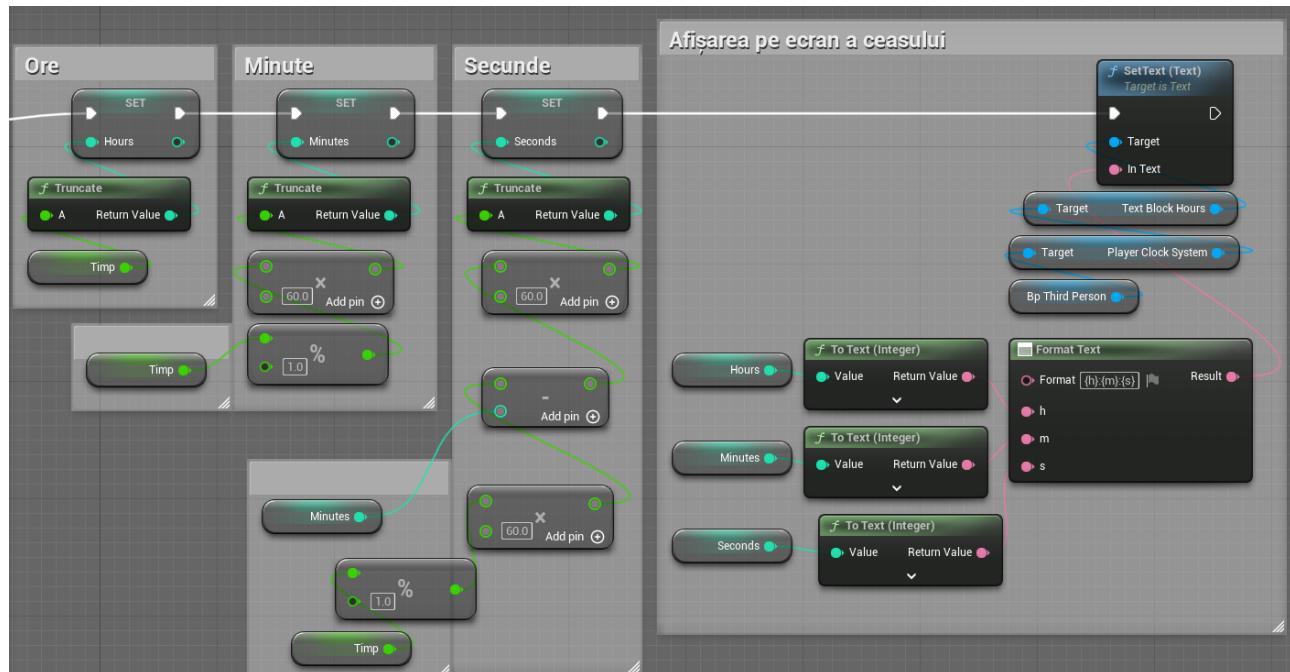
Momentul de zi/noapte



Figură 5.86 – Reprezentarea vizuală a rotației surselor de lumină

Gestionarea timpului din joc

Gestionarea timpului reprezintă o componentă adițional-importanță, responsabilă de contorizarea timpului curent și conversie acestuia într-un sistem de simulare în timp real al trecerii orelor, minutelor și secundelor din joc realizând astfel funcționalitatea unui ceas intern de tip format [hh:mm:ss]. [Figură 5.87]



Figură 5.87 – Gestionarea orelor, minutelor și a secundelor (Blueprint)

- **hours** – variabila de tip integer responsabilă în contorizarea orelor.

- **minutes** – variabilă de tip integer responsabilă pentru contorizarea minutelor.
- **seconds** – variabilă de tip integer responsabilă pentru contorizarea secundelor.
- **Truncate** – nod de conversie al valorilor de tip float în valori de tip integer.

Pentru a realiza această funcționalitate de simulare în timp real al unui ceas în mediul de joc virtual, am utilizat timpul curent (delta time) în felul următor:

Prin nodul **Truncate** am realizat operația **de extragere a părții întregi** a valorii de tip float convertind astfel valorilor în integer, determinând astfel **variabila responsabilă de contorizare a orelor. Ora = Truncate(Timp)**

Pentru determinarea minutelelor, am folosit următoarea formulă:

$$\text{Minute} = \text{Truncate}((\text{Timp} \% 1) * 60)$$

Aceasta formulă extrage partea zecimală a valorii de timp și o înmulțește cu 60 pentru a determina **dupa trunchiere rezultatului contorizarea minutelor**.

Iar pentru determinarea secundelor, am folosit următoarea formulă:

$$\text{Secunde} = (((\text{Timp} \% 1) * 60) - \text{minute}) * 60$$

Formula calculează mai întai partea zecimală a timpului pentru minute, apoi extrage la rândul ei partea zecimală a minutelor și o **convertește în secunde determinând astfel dupa trunchiere rezultatului contorizarea secundelor. [Listare 5.27]**

```

La fiecare moment de timp
    timp = (deltaSeconds / timeDelay) + timp
    IF(timp >= 24), THEN
        |      timp = 0
    ELSE
        |      Actualizăm rotația soarelui(((timp - 19) + 24) / 24) * 360)
        |      Actualizăm rotația lunii(((timp - 19) + 24) / 24) * 360) + 180
        |
        |      hours = Truncate(timp)
        |      minutes = Truncate((timp \% 1) * 60)
        |      seconds = Truncate(((timp \% 1) * 60) - minute) * 60)
        |      Actualizăm textul de pe ecran responsabil de contorizarea
        |      timpului.
    END IF

```

Exemplu:

1) Considerăm Timp = 22.76

Ora = Truncate(22.76)

Ora = 22

Minute = Truncate((22.76 % 1) * 60)

$$= \text{Truncate}(0.76 * 60)$$

$$= \text{Truncate}(45.6)$$

Minute = 45

Secunde = Truncate(((22.76 % 1) * 60) - 45) * 60

$$= \text{Truncate}(((0.76 * 60) - 45) * 60)$$

$$= \text{Truncate}((45.6 - 45) * 60)$$

$$= \text{Truncate}(0.6 * 60)$$

Secunde = 36

Deci, Timp = 22.76 (float) va fi echivalent în sistemul de contorizare al timpului cu 22:45:36.

2) Considerăm Timp = 9.99

Ora = Truncate(9.99)

Ora = 9

Minute = Truncate((9.99 % 1) * 60)

$$= \text{Truncate}(0.99 * 60)$$

$$= \text{Truncate}(59.4)$$

Minute = 59

Secunde = Truncate(((9.99 % 1) * 59) - 45) * 60

$$= \text{Truncate}(((0.99 * 60) - 59) * 60)$$

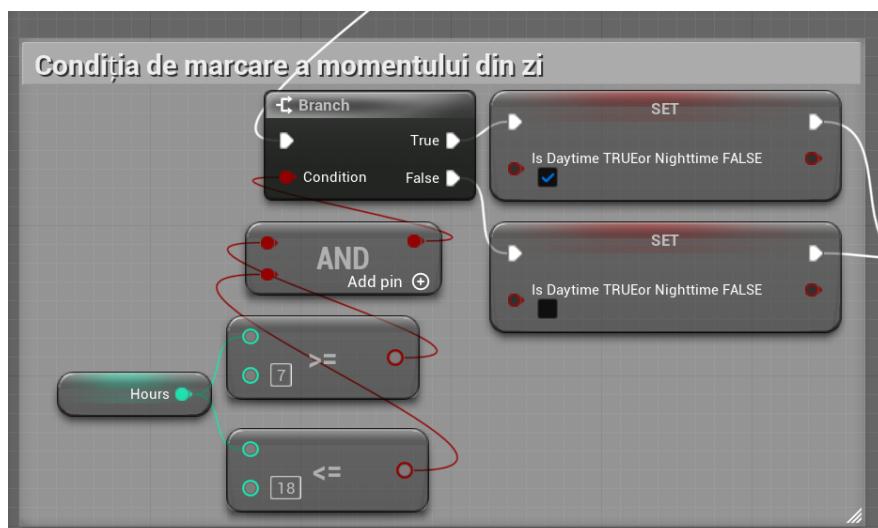
$$= \text{Truncate}((59.4 - 59) * 60)$$

$$= \text{Truncate}(0.4 * 60)$$

Secunde = 4

Deci, Timp = 9.99 (float) va fi echivalent în sistemul de contorizare al timpului cu 09:59:04.

Tot aici avem partea care se ocupă cu determinarea momentului actual din zi, prin logica următoare:



Figură 5.88 – Marcarea momentului de timp zi/noapte (Blueprint)

```

La fiecare moment de timp
  IF(hours >= 7 && <= 18), THEN
    |      isDaytimeTRUEorNightTimeFALSE = True
  ELSE
    |      isDaytimeTRUEorNightTimeFALSE = False
END IF

```

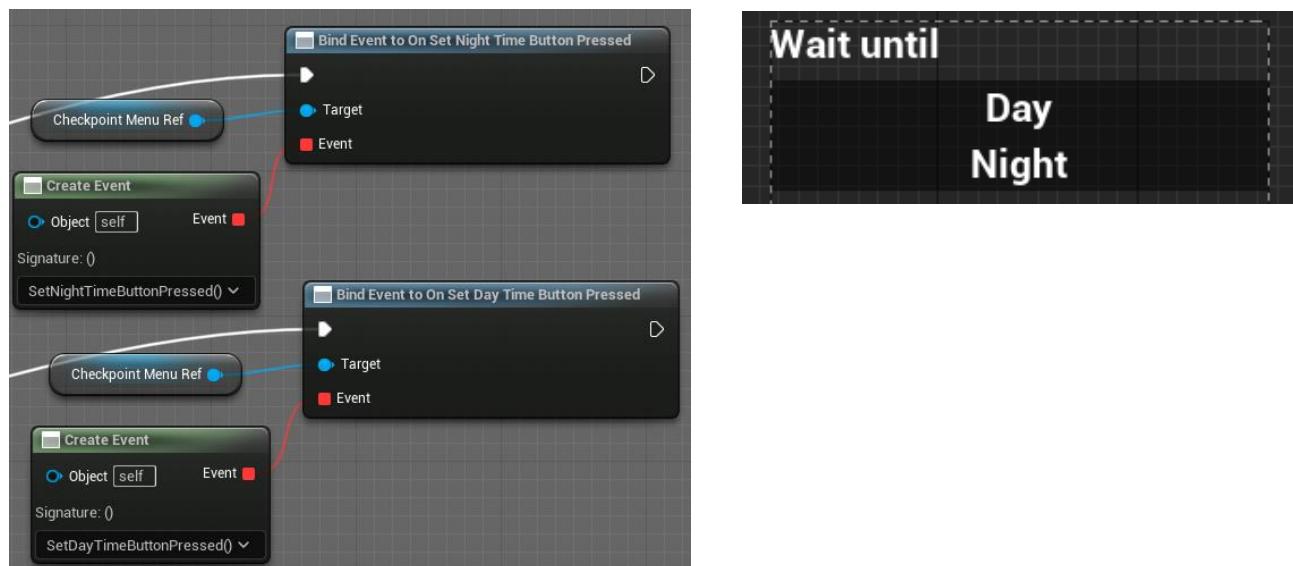
- **isDaytimeTRUEorNightTimeFALSE**: variabilă de tip boolean responsabilă de determinarea stării actuale a ciclului de zi/noapte.

De ce am ales 18 și nu 19? Datorită realizării unei tranziții mai fine la apus și răsărit pentru ca trecerea dintre cele două surse de lumină să nu fie una bruscă.

Prin urmare am reprezentat sub-sistemul de ore - zi/noapte. Care permite schimbul trecerii de timp dinamic. Dar și posibilitatea de comutare manuală, prin intermediul butoanelor de:

- setDayTimeButtonPressed
- setNightTimeButtonPressed

Prin intermediul evenimentului **onClick()**, folosit ca și bind de legătură care acționează în momentul apăsării a funcțiilor acestora.



Figură 5.89 – Bind-uri buton la apăsare pentru schimbul de timp în joc



Figură 5.90 – Funcțiile de setare zi/noapte (Blueprint)

La apăsarea butonului de Day(zii)

Începem o tranziție de fade a camerei

timpNou = 12

Apelăm funcția **adjustDayNightTime** din cadrul sub-sistemului zi/noapte

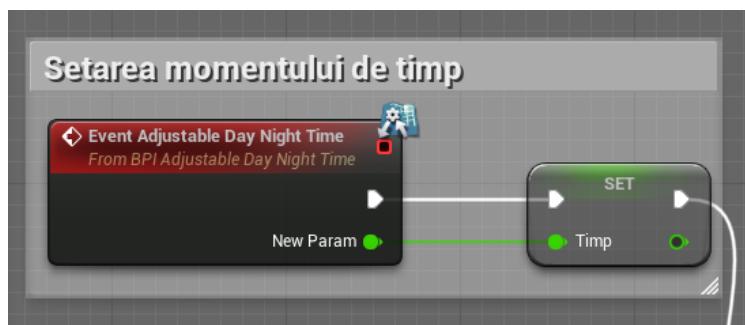
La apăsarea butonului de Night(noapte)

Începem o tranziție de fade a camerei

timpNou = 20

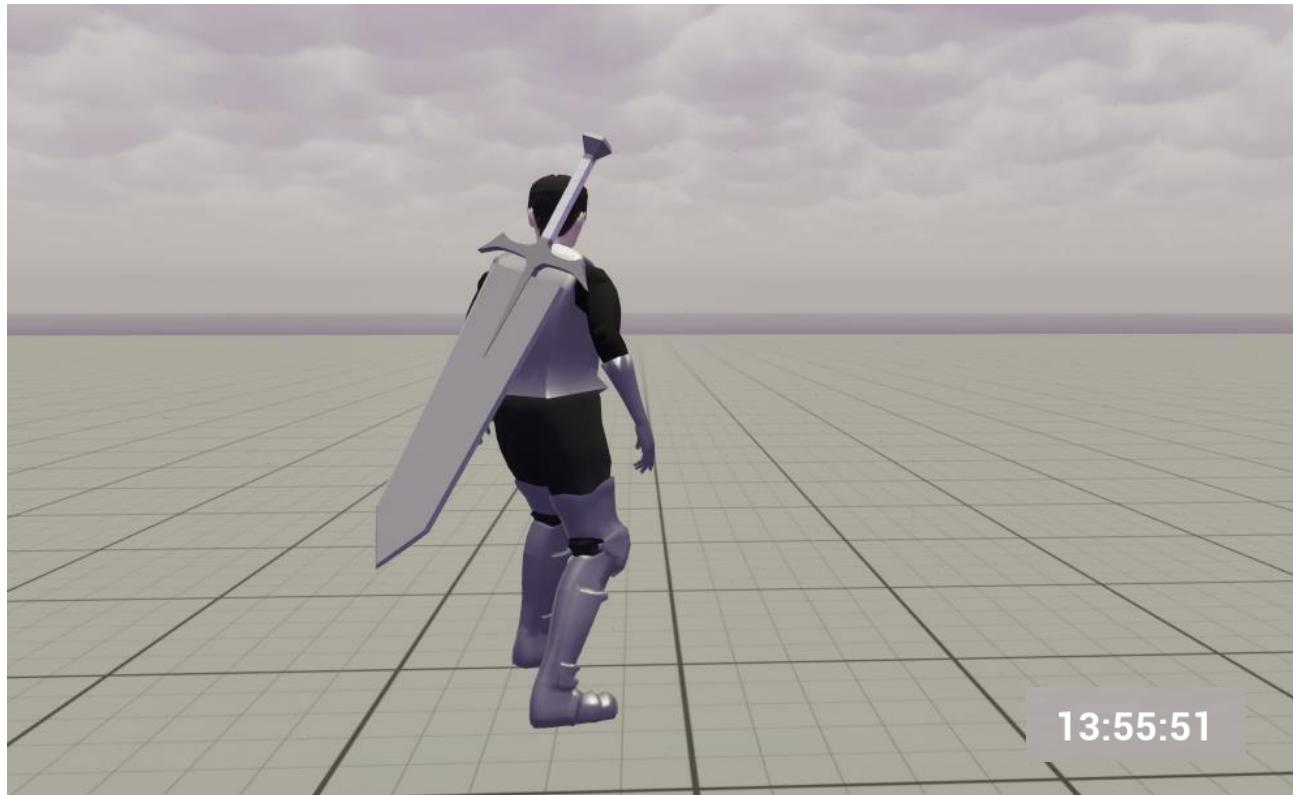
apelăm funcția **adjustDayNightTime** din cadrul sub-sistemului zi/noapte

Prin cele două funcții, **setDayTimeButtonPressed** și **setNightTimeButtonPressed** setăm timpul din cadrul sub-sistemului de zi/noapte printr-o interfață Blueprint cu funcția „**AdjustDayNightTime**”.



Figură 5.91 – Interfața de comunicare pentru setare zi/noapte (Blueprint)

Reprezentarea vizuală a scenei pe timp de zi



Figură 5.92 – Reprezentarea vizuală a scenei pe timp de zi

Reprezentarea vizuală a scenei pe timp de noapte



Figură 5.93 – Reprezentarea vizuală a scenei pe timp de noapte

5.5.2. Sub-sistemul de anotimpuri

Sub-sistemul a fost realizat pentru a simula cele patru anotimpuri – primăvară, vară, toamnă și iarnă, oferind jucătorului posibilitatea de a le schimba manual. Această funcționalitate influențează aspectul vizual al mediului înconjurător în funcție de preferințele jucătorului, contribuind astfel la îmbogătirea experienței de joc.

Modul de crearea al Materialului Automat (AutoMesh)

Pentru realizarea acestuia am folosit 6 funcții de materiale (Material Function)

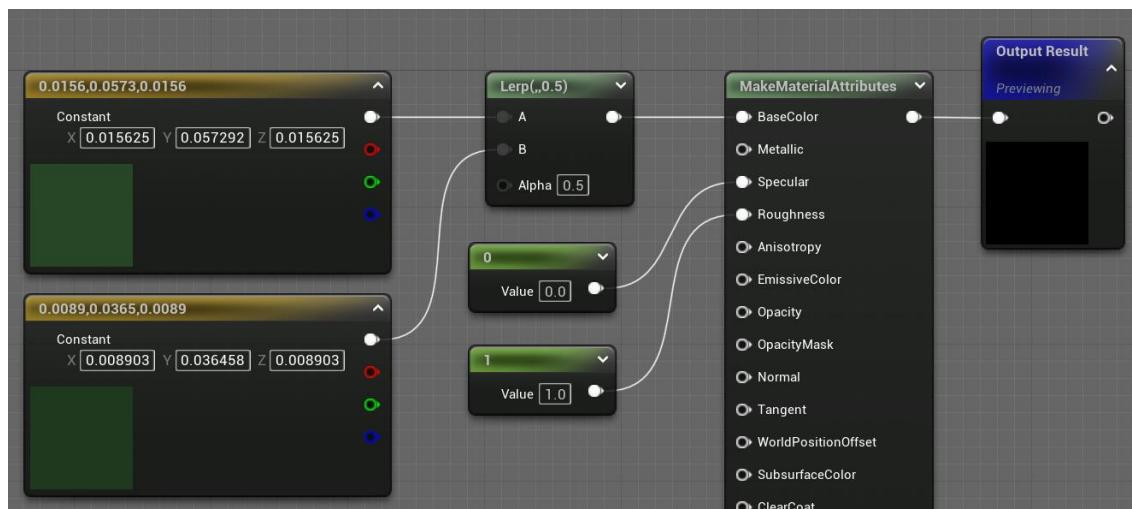
- 4 funcții pentru texturile de anotimpuri: primăvară, vară, toamnă și iarnă.
- 1 funcție pentru textura de piatră.
- 1 Material Manager



Figură 5.94 – Lista funcțiilor de funcții ale materialelor

Crearea funcției de culoare material

Toate cele 4 materiale de anotimpuri sunt realizate prin această logică. Singura diferență pe care o au este culoare. [Figură 5.95]



Figură 5.95 – Determinarea culorii unui material funcție (Blueprint)

BaseColor: determină culoarea texturii având asociat 2 vectori de tip **Constant 3 Vector** care controlează valorile RGB ale unei culori. Rezultatul celor 2 vectori de culoare este conectat la un **Lerp (Linear Interpolation)** care prin canalul Alpha al acestuia se poate determina factorul de amestec:

- 0 – va returna culoarea A fară modificări
- 1 – va returna culoare B fară modificări
- 0.5 – va returna un amestec între cele 2 culori A și B rezultând mijlocul. (50% 50%)

Specular: controlează intensitatea reflectării unei suprafete.

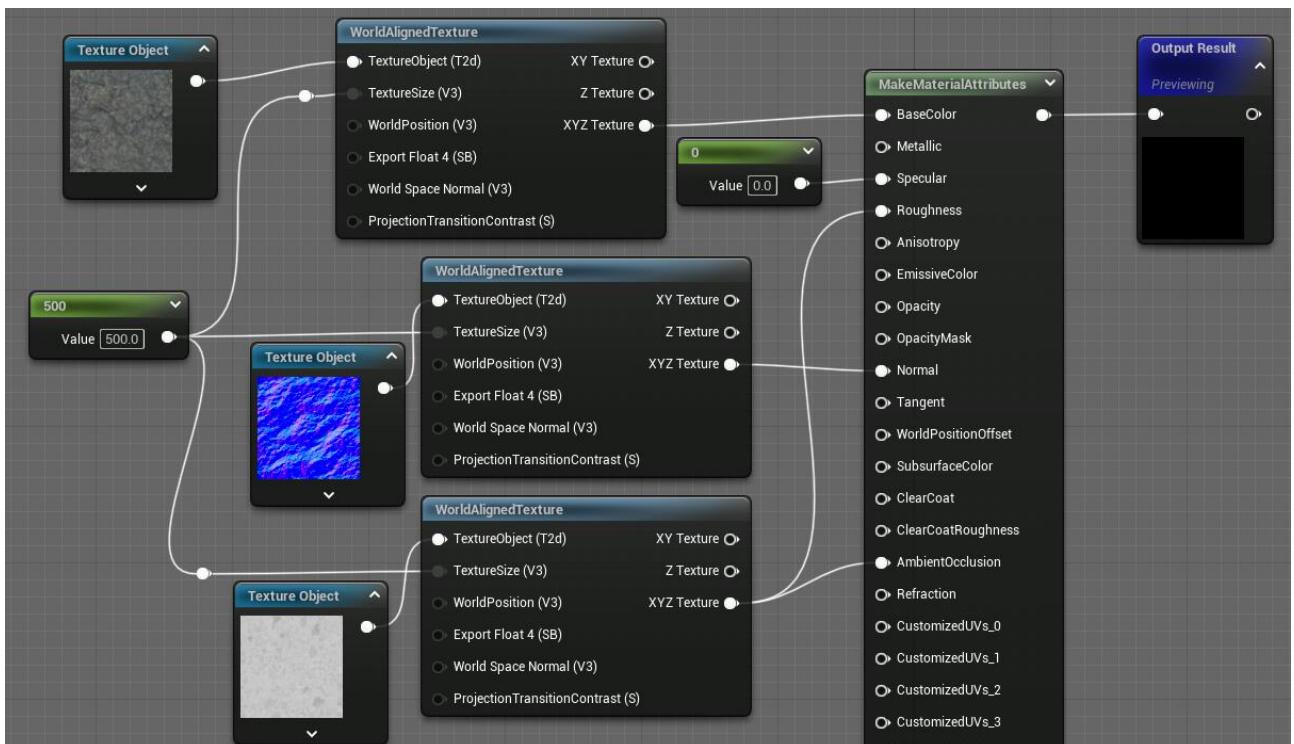
- 0: suprafata nu reflectă lumina
- 1: suprafata reflectă lumina (lucioasa)

Roughness: controleaza netezimea unei suprafete.

- 1: reflexii clare
- 0: reflexii difuze/ inexistente

Aceste trei atribute de BaseColor, Specular și Roughness vor controla textura material de anotimp.

Crearea funcției de textură material



Figură 5.96 – Determinarea culorii unui material funcție (Blueprint)

Pentru a crea texturea de piatră:

BaseColor: culoarea texturii de data aceasta va fi însuși o textură.

Specular: controleaza intensitatea reflectarii suprafetei a texturii

- 0 - suprafața nu reflectă lumină
- 1 - suprafața reflectă lumină

Normals: determină modul în care lumina interacționează cu suprafata texturii, folosind textura de normals a aceasta.

Roughness: controlează netezimea texturii, determinată tot de textura de roughness a acesteia.

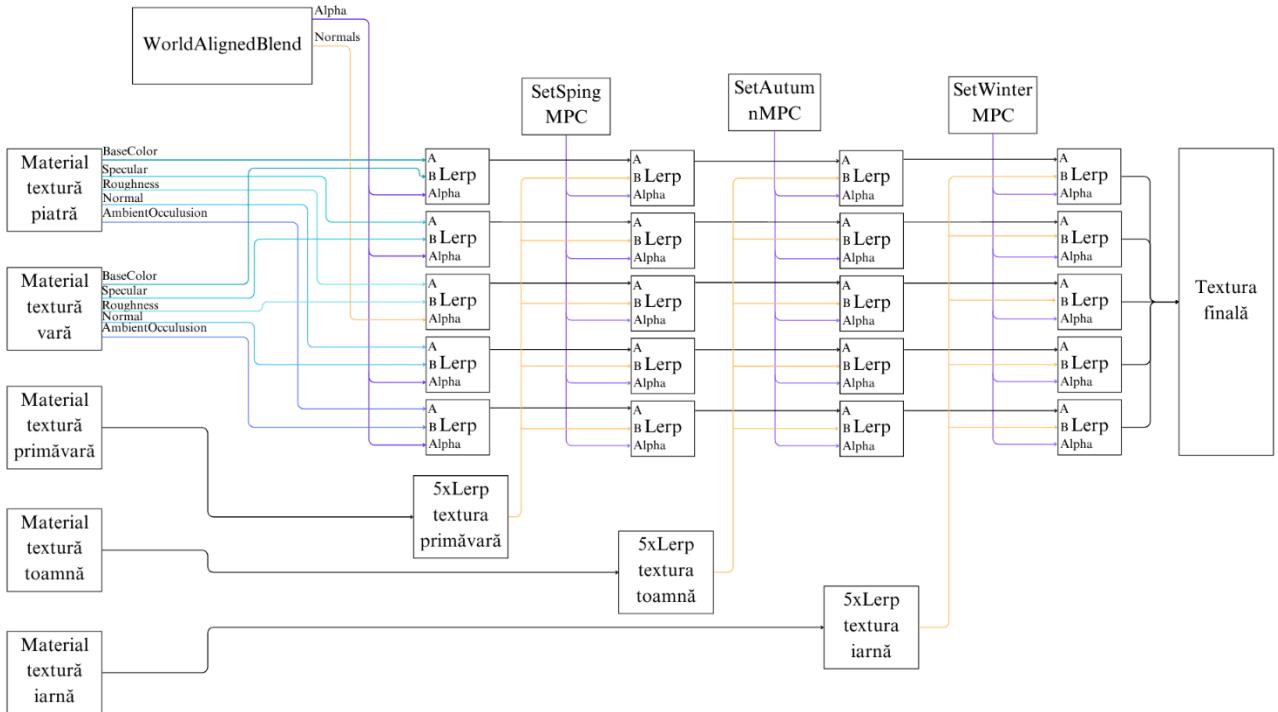
AmbientOcclusion: simulează modul prin care lumina este blocată în spațiile mai stramă sau în zonele în care obiectele sunt mai aproape unele de altele. La fel vom folosi textura acesteia specială pentru ambient.

WorldAlignedTexture: este un nod în Unreal Engine (Blueprints sau Material Editor) care proiectează o textură în spațiul lumii, nu pe coordonatele UV locale ale modelului 3D.

Crearea funcției de material automat

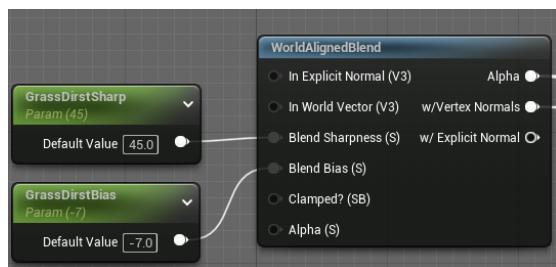
După ce am realizat cele cinci funcții de materiale. Le vom include pe acestea într-un material function main care va controla logica de selecție a texturilor de materiale ale anotimpurilor. [Figură 5.95]

Schema bloc a materialului automat



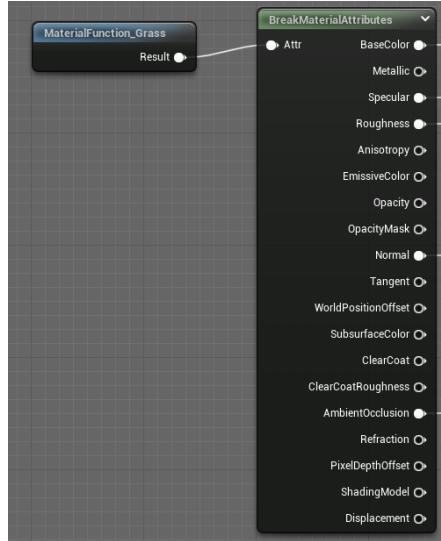
Figură 5.97 – Schema bloc a materialului automat a texturii de iarba

Acest prim grup de lerpuri: **GrassDirstSharp** și **GrassDirstBias** (sunt folosiți pentru a combina cele 2 texturi de iarba și piatră)



Figură 5.98 – Nodul de blend al texturilor

- **GrassDirstSharp:** determină care material predomină una față de celălaltă.
- **GrassDirstBias:** este folosit pentru a seta de la ce înălțime textura începe să se modifice.

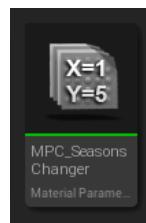


Figură 5.99 – Nodul de separare parametrii

BreakMaterialAttributes este un nod care separă funcția în mai mulți pini pentru a extrage informația curentă din aceștia. În cazul nostru vom folosi doar BaseColor, Specular, Roughness, Normal și AmbientOcculusion. **Fiecare atribut cu Lerp-ul său.** [Figură 5.99]

Acum că avem cele patru texturi, mai precis: primăvară, vară, toamnă, iarnă cu atrbutele texturii de iarba și piatră. Nu mai avem decat să schimbăm textura ierbii.

Ne vom folosi din nou de Lerp, numai că de data aceasta o să folosim un **MPC (Material Parameter Collection)** ca și validare alpha. [Figură 5.100]



Figură 5.100 – Material Parameter Collection

Creăm trei variabile: **SetSpringMPC**, **SetAutumnMPC** și **SetWinterMPC** setate inițial cu valoare 0.

De ce nu există și **SetSummerMPC**?, deoarece acesta este folosit ca **default**.

Parametrii de material vor seta doar atrbutele texturii de anotimp nu și cea de piatră. Textura de piatra va fi materialul principal iar iarba complement care modifică textura de anotimp.

Ideea este urmatoarea, aceste variabile Set... vor aciona ca și canale Alpha pentru a valida output-ul final al unui lerp. Practic toate lerp-urile vor fi într-un lanț conectate (în funcție de atrbutul fiecaruia) iar valoarea set-urilor va determina anotimpul curent. [Figură 5.101]



Figură 5.101 – Nodul Lerp

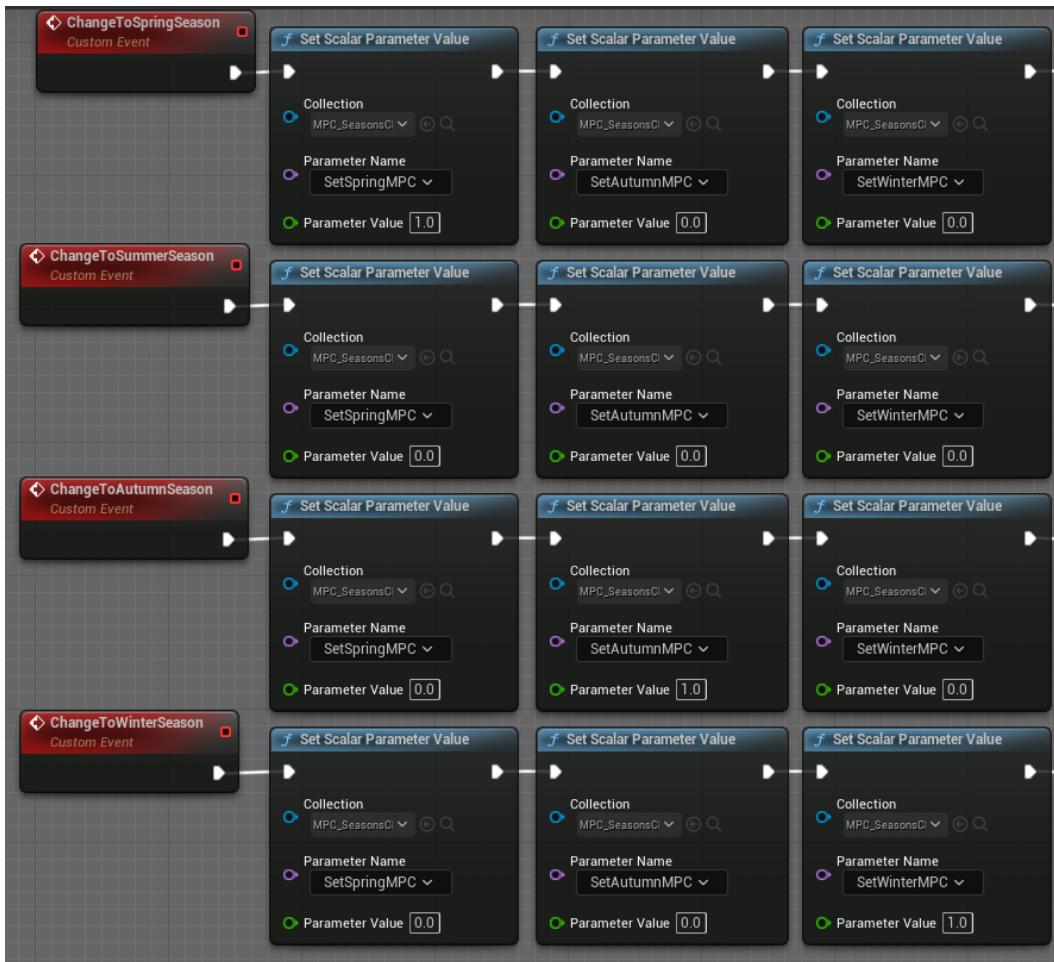
Acum că avem managerul de materile care conține logica pentru determinarea texturii unui anotimp, implementarea va fi realizată în felul următor.



Figură 5.102 – Butoanele pentru schimbul de anotimpuri

Butoanele interfeței UI vor fi apelate în momentul evenimentului `onClick()` printr-un event dispatcher din cadrul interfeței UI, iar bind-ul acestora vor face posibilă comutarea între anotimpuri.

- **Butonul Spring** – se ocupă cu setarea materialului de textură a primăverii prin custom evenut `changeToSpringSeason`.
- **Butonul Summer** – se ocupă cu setarea materialului de textură a verii prin custom evenut `changeToSummerSeason`.
- **Butonul Autumn** – se ocupă cu setarea materialului de textură a toamnei prin custom evenut `changeToAutumnSeason`.
- **Butonul Winter** – se ocupă cu setarea materialului de textură a iernii prin custom evenut `changeToWinterSeason`.



Figură 5.103 – Funcțiile pentru anotimpuri (Blueprint)

La apăsarea butonului Spring, apelăm changeToSpringSeason (primăvară)

```
setSpringMPC = 1  
setAutumnMPC = 0  
setWinterMPC = 0
```

La apăsarea butonului Summer, apelăm changeToSummerSeason (vară)

```
setSpringMPC = 0  
setAutumnMPC = 0  
setWinterMPC = 0
```

La apăsarea butonului Autumn, apelăm changeToAutumnSeason (toamnă)

```
setSpringMPC = 0  
setAutumnMPC = 1  
setWinterMPC = 0
```

La apăsarea butonului Winter, apelăm changeToWinterSeason (iarnă)

```
setSpringMPC = 0  
setAutumnMPC = 0  
setWinterMPC = 1
```

Prin această logică de selecție o să determinăm textura anotimpului care va fi selectată în urma combinațiilor de parametrii ale materialelor MPC din cadrul **Managerului de materiale**.

Tabelul de combinații pentru determinarea de anotimp

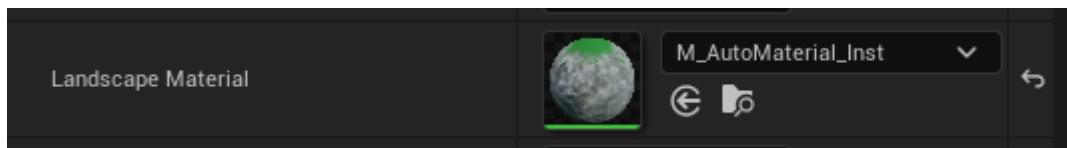
WinterMPC	AutumnMPC	SpringMPC	OUTPUT
0	0	0	Vară
0	0	1	Primăvară
0	1	0	Toamnă
1	0	0	Iarnă

Tabel 5.6 – Tabelul combinațiilor pentru determinarea de anotimp

Doar o singură ieșire va fi posibil activă în momentul selecției.

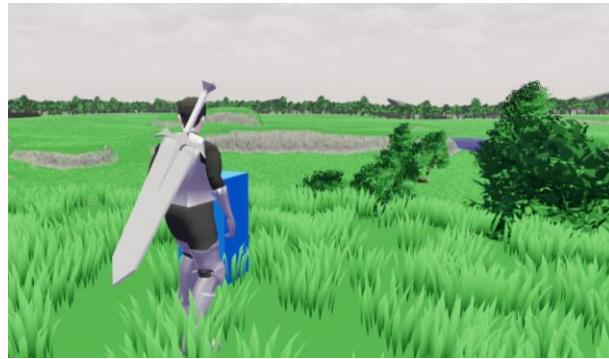
Textura materialului manager este atribuită podelei de joc care o să alcătuiască Landscape-ul.

Setată materialul din proprietățile acestuia, la secțiunea de **Landscape material**. [Figură 5.104]



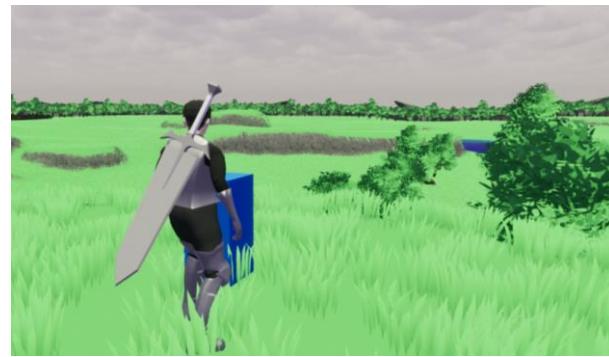
Figură 5.104 – Setarea materialului landscape

Reprezentarea vizuală a anotimpului de primăvară



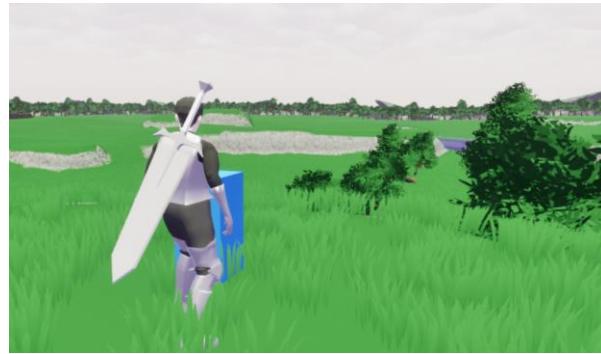
Figură 5.105 – Rerezentarea vizuală a anotimpului de primăvară

Reprezentarea vizuală a anotimpului de vară



Figură 5.106 – Rerezentarea vizuală a anotimpului de vară

Reprezentarea vizuală a anotimpului de toamnă



Figură 5.107 – Rerezentarea vizuală a anotimpului de

Reprezentarea vizuală a anotimpului de iarnă



Figură 5.108 – Rerezentarea vizuală a anotimpului de iarnă

5.5.3. Sub-sistemul de fenomene meteorologice

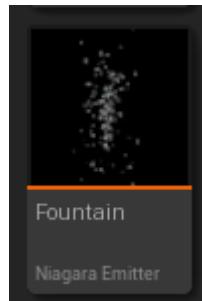
Sub-sistemul a fost realizat pentru **simularea fenomenelor din natură** cum ar fi: **ploaia, ninsoarea sau ceața**. Oferind jucătorului posibilitatea de a le schimba manual. Această funcționalitate influențează aspectul vizual al mediului înconjurător în funcție de preferințele jucătorului, contribuind astfel la îmbogătirea experienței de joc.

Modul de creare al unui sistem de particule Niagara

Pentru a crea vizual simularea de efecte precum ploaie sau ninsoare, Unreal Engine ne pune la dispoziție sistemul **Niagara Effects**. [31]

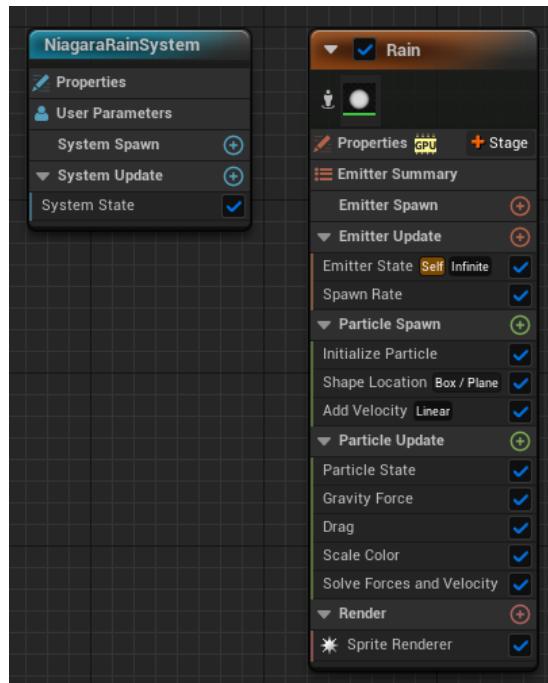
Acest tool având o varietate de şablonane template din care putem alege pentru a realiza un sistem de particule cu o anumită funcționalitate.

Cele două efecte de particule ploaie și ninsoare vor fi realizate prin template-ul de **Fountain Niagara Emitter**. [Figură 5.109]



Figură 5.109 – Fountain Niagara Emitter

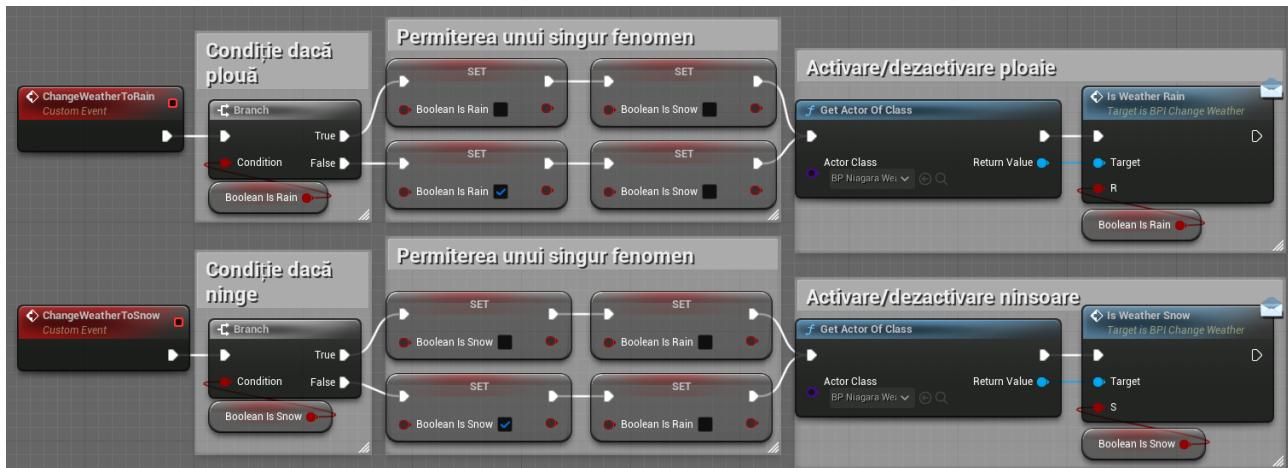
În acestă interfață o să controlăm diferitele proprietăți ale unui efect, prin modificarea parametrilor cum ar fi: rata de apariție a particulelor, dimensiunea particulelor, forma particulelor, viteza de deplasare a acestora, direcția de deplasare, gravitația, forța de tragere, pentru a realiza efectul dorit. [Figură 5.110]



Figură 5.110 – Fountain Niagara Emitter Parametrii



Figură 5.111 – Fountain Niagara Emitter Parametrii



Figură 5.112 – Funcțiile de setare a vremii (Blueprint)

La apăsarea butonului Rain, apelăm changeWeatherToRain()

```

IF(booleanIsRain == True), THEN
|   booleanIsRain = False
|   booleanIsSnow = False
|   isWeatherRain()
ELSE
|   booleanIsRain = True
|   booleanIsSnow = False
|   isWeatherRain()
END IF

```

La apăsarea butonului Snow, apelăm changeWeatherToSnow()

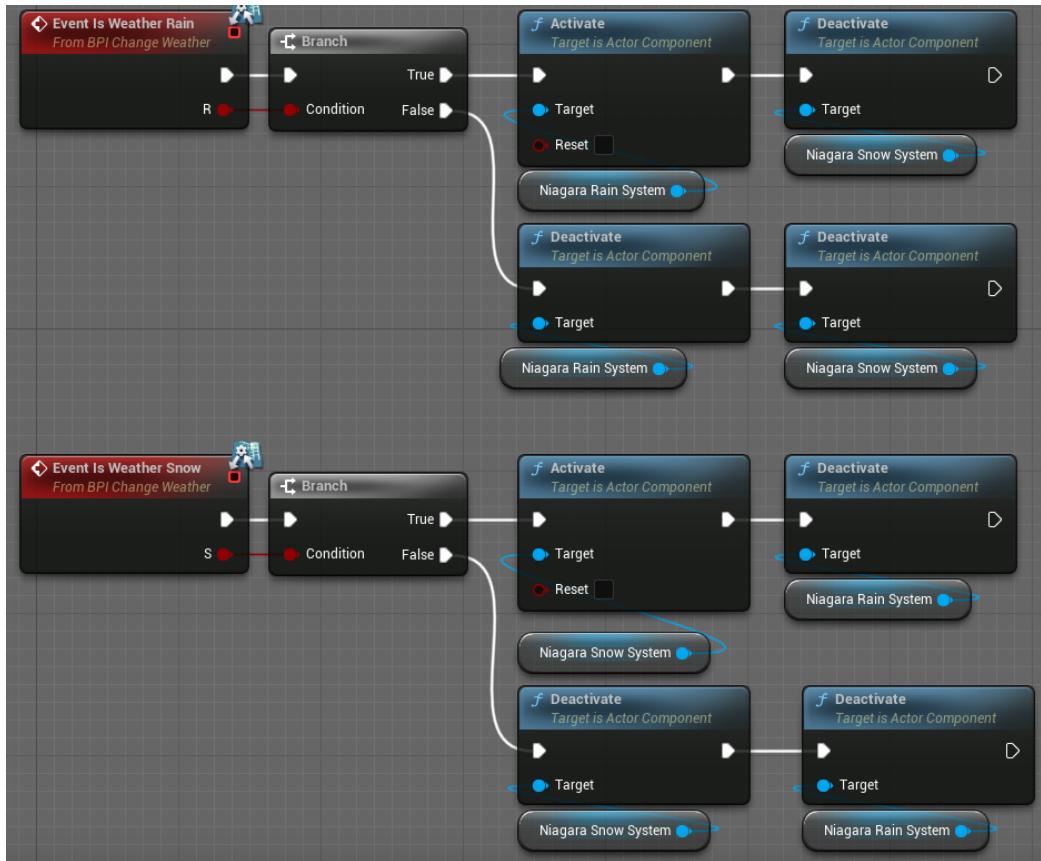
```

IF(booleanIsSnow == True), THEN
|   booleanIsSnow = False
|   booleanIsRain = False
|   isWeatherSnow()
ELSE
|   booleanIsSnow = True
|   booleanIsRain = False
|   isWeatherSnow()
END IF

```

- **booleanIsRain** – variabilă de tip boolean folosită pentru a indica fenomenul de ploaie activ.
 - **booleanIsSnow** – variabilă de tip boolean folosită pentru a indica fenomenul de ninsoare activ.
 - **isWeatherRain()** – funcția conține logica de activare și dezactivare a sistemului de particule niagara aferent fenomenului de ploaie.
 - **isWeatherSnow()** – funcția conține logica de activare și dezactivare a sistemului de particule niagara aferent fenomenului de ninsoare.
- Prin acest mod permitem doar unui singur fenomen să fie activ în scenă.

Activarea și dezactivarea fenomenelor



Figură 5.113 – Activarea și dezactivarea fenomenelor (Blueprint)

```
Functia changeWeatherToRain()
  IF(R = True), THEN
    | Activăm sistemul de particule pentru ploaie
    | Dezactivăm sistemul de particule pentru ninsoare
  ELSE
    | Dezactivăm sistemul de particule pentru ploaie
    | Dezactivăm sistemul de particule pentru ninsoare
END IF
```

```
Functia changeWeatherToSnow()
  IF(S == True), THEN
    | Activăm sistemul de particule pentru ninsoare
    | Dezactivăm sistemul de particule pentru ploaie
  ELSE
    | Activăm sistemul de particule pentru ninsoare
    | Dezactivăm sistemul de particule pentru ploaie
END IF
```

Rain (R)	Snow (S)	OUTPUT
True	False	Ploie activă
False	True	Ninsoare activă
False	False	Fenomene dezactivate

Tabel 5.7 – Tabelul combinațiilor pentru determinarea de fenomene meteorologice (ploaie/ ninsoare)

Reprezentarea vizuală a fenomenului de ploaie

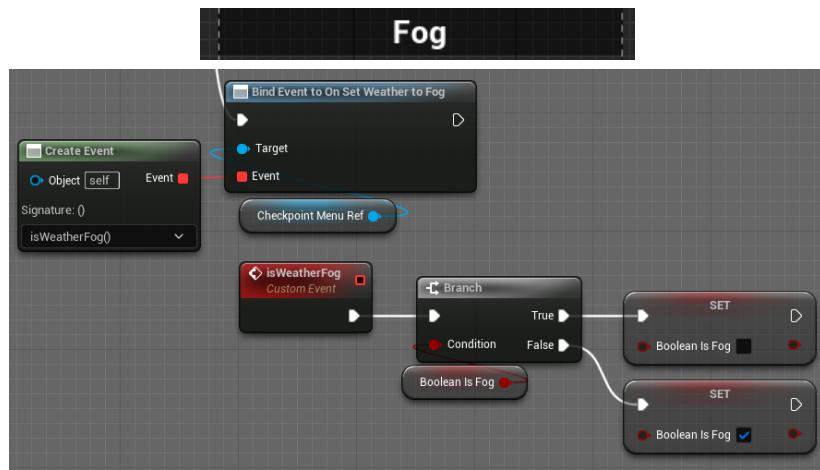


Figură 5.114 – Reprezentarea vizuală a fenomenului de ploaie

Reprezentarea vizuală a fenomenului de ninsoare

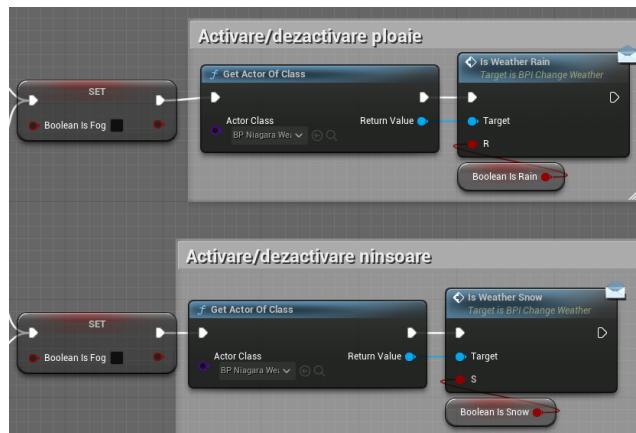


Figură 5.115 – Reprezentarea vizuală a fenomenului de ninsoare



Figură 5.116 – Butonul de ceată

Butonul Fog la apăsare va alterna între cele două stări posibile și va seta variabila **booleanIsFog** pentru permiterea de ceată în scena jocului. De va fi necesară modificarea logicii a figurii [Figură 5.117] pentru a permite resetarea fenomenului de ceată.



Figură 5.117 – Resetarea de ceată (Blueprint)

```

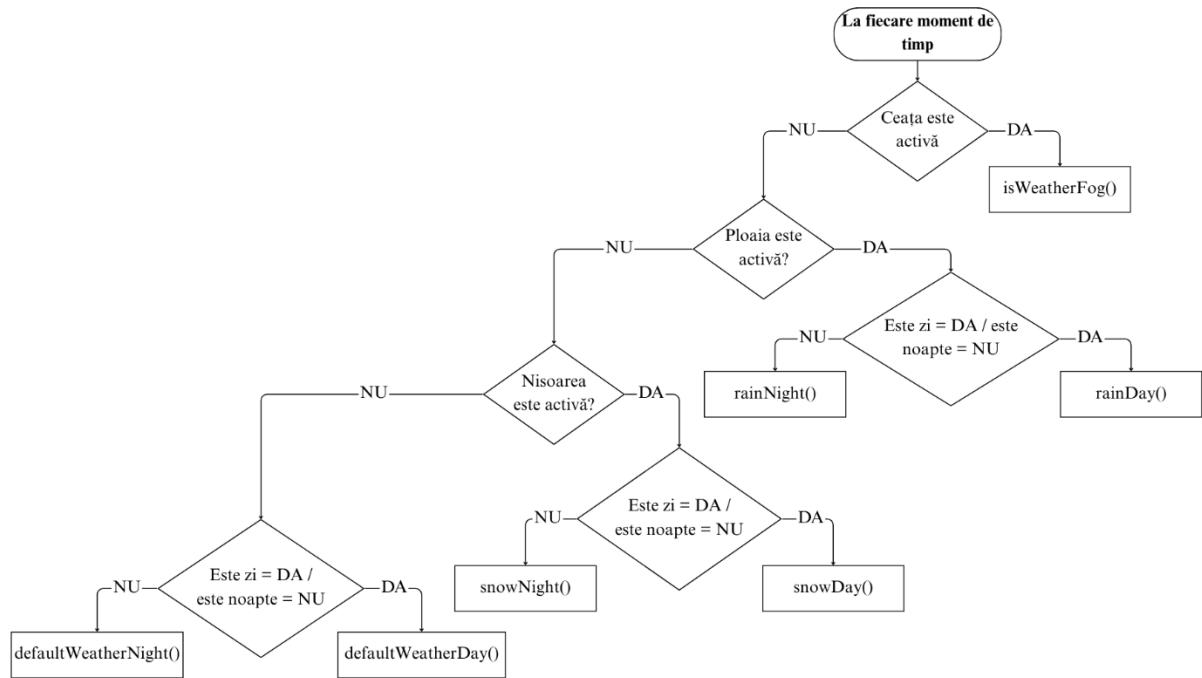
IF(booleanIsRain == True), THEN
| ..
|   booleanIsFog = False
|   isWeatherRain()
ELSE
| ..
|   booleanIsFog = False
|   isWeatherRain()
END IF

IF(booleanIsSnow == True), THEN
| ..
|   booleanIsFog = False
|   isWeatherSnow()
ELSE
| ..
|   booleanIsFog = False
|   isWeatherSnow()
END IF

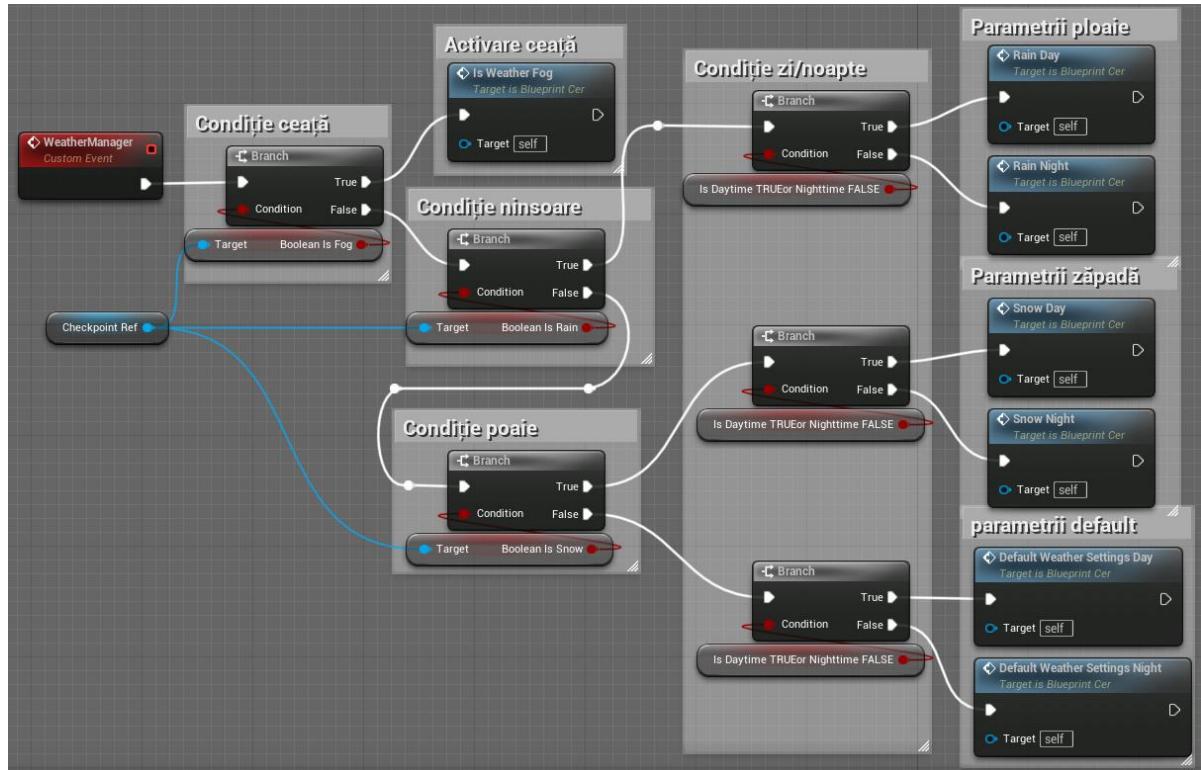
```

Managerul de vreme

Managerul de vreme reprezintă logica de selecție pe care acesta o îndeplinește pentru a seta condițiile meteorologice în funcție de starea actuală a jocului. Aceasta analizează variabile precum momentul zilei (zi/noapte), tipul de vreme selectată (ceată, ploaie, ninsoare) și stabilește parametrii corespunzători pentru fiecare situație, asigurând astfel o tranziție realistă și o atmosferă dinamică în cadrul jocului.



Figură 5.119 – Organigramă generală a managerului de vreme



Figură 5.118 – Funcția managerului de vreme (Blueprint)

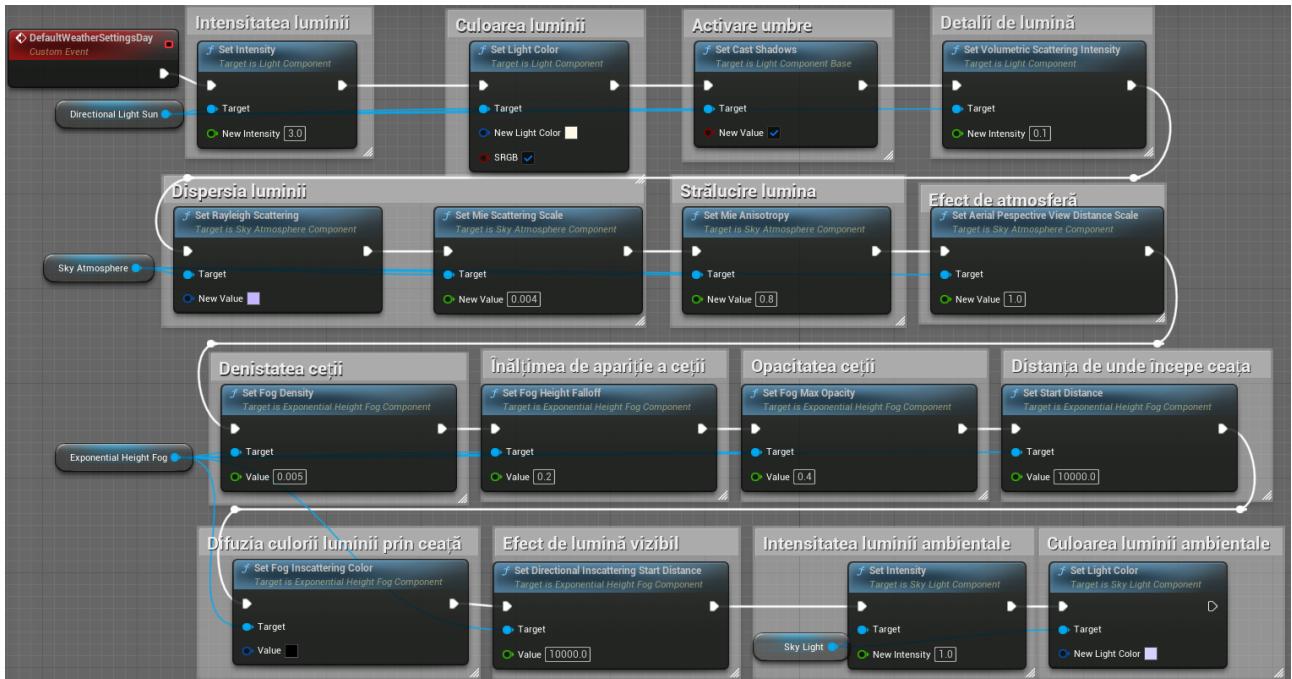
```

La fiecare moment de timp
    IF(booleanIsFog == True), THEN
        | isWeatherFog()
    ELSE
        | IF(booleanIsRain == True), THEN
            | | IF(isDayTimeTRUEorNightTimeFALSE == True), THEN
            | | | rainDay()
            | | ELSE
            | | | rainNight()
        | END IF
    ELSE
        | IF(booleanIsSnow == True), THEN
            | | IF(isDayTimeTRUEorNightTimeFALSE == True), THEN
            | | | snowDay()
            | | ELSE
            | | | nowNight()
        | END IF
    ELSE
        | | IF(isDayTimeTRUEorNightTimeFALSE == True), THEN
        | | | defaultWeatherSettingsDay()
        | | ELSE
        | | | defaultWeatherSettingsNight()
    | END IF
| END IF

```

- **booleanIsFog** – variabilă de tip boolean folosită pentru condiția de ceată activă.
- **booleanIsRain** – variabilă de tip boolean folosită pentru condiția de ploaie activă.
- **booleanIsSnow** – variabilă de tip boolean folosită pentru condiția de ninsoare activă.
- **isDayTimeTRUEorNightTimeFalse** – variabilă de tip boolean folosită pentru a determina momentul de timp actual, True – este zi, False – este noapte)
- **isWeatherFog()** – funcție care setează efectul de ceată în scena jocului.
- **rainDay()** – funcție utilizată pentru setarea de ploaie pe timp de zi.
- **rainNight()** – funcție utilizată pentru setarea de ploaie pe timp de noapte.
- **snowDay()** – funcție utilizată pentru setarea de ninsoare pe timp de zi.
- **snowNight()** – funcție utilizată pentru setarea de ninsoare pe timp de noapte.
- **defaultWeatherSettingsDay()** – funcție default utilizată pentru setarea de zi senină.
- **defaultWeatherSettingsNight()** – funcție default utilizată pentru setarea de noapte senină.
- **checkpointRef** – referința punctului de acces.

Funcțiile managerului de vreme sunt esențiale modului de reprezentare vizual în scenă, deoarece acestea conțin parametrii componentelor de mediu înconjurător. **Pentru a controla ambiența din scenă**, astfel:



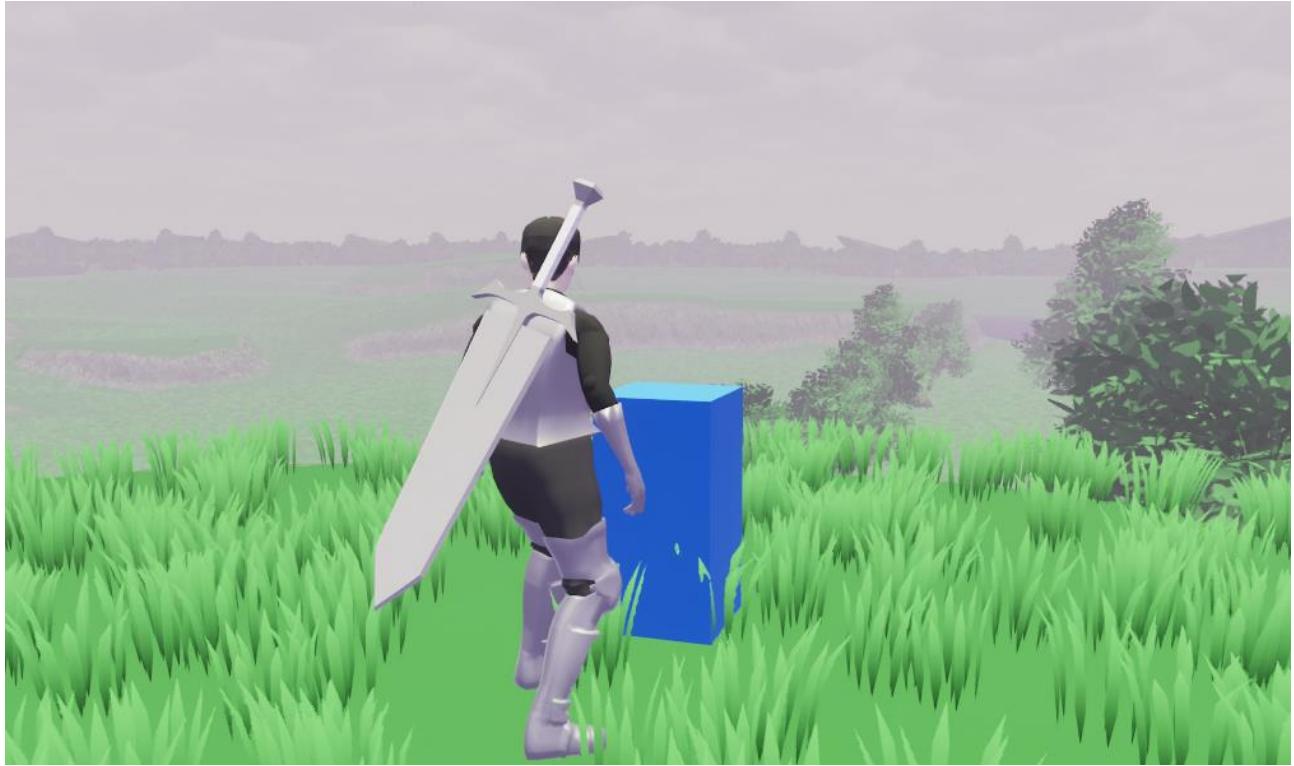
Figură 5.120 – Exemplu funcție de ambiență a scenei

De asemenea celelalte funcții vor fi identice ca structură, dar cu valori potrivite acestora. **Singura diferență o fac funcțiile de noapte**, unde sursa de lumină trebuie schimbată cu cea a lunii.

Iar **isWeatherFog()** va seta doar parametrii funcțiilor care țin de componența ceață și anume:

- **setFogDensity** – funcție ce controlează densitatea ceții.
- **setFogHeightFalloff** – funcția determină densitatea ceții în funcție de altitudine.
- **setStartDistance** – funcția determină de la ce distanță începe ceață.
- **setFogMaxOpacity** – funcția determină opacitatea ceții
- **setSecondFogDensity** – funcția controlează densitatea celei de-al doilea strat de ceață.
- **setSecondHeightOffset** - funcția determină de la ce distanță începe al doilea strat de ceață.
- **DirectionalLightSun, SkyAtmosphere, ExponentialFog, Skylight** – componente care alcăuiesc cerul în joc.

Reprezentarea vizuală a fenomenului de ceață pe timp de zi



Figură 5.121 – Reprezentarea vizuală a fenomenului de ceață pe timp de zi

Reprezentarea vizuală a fenomenului de ceață pe timp de noapte



Figură 5.122 – Reprezentarea vizuală a fenomenului de ceață pe timp de noapte

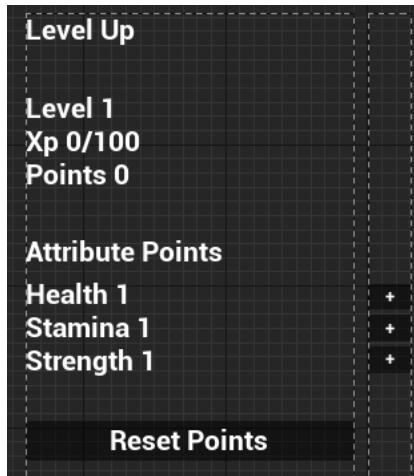
Astfel, prin implementarea acestui sistem manager de vreme, am realizat o variată gamă de combinații între anotimpuri, momente ale zilei și fenomene meteorologice.

Anotimp	Moment al zilei	Fenomen meteologic	Rezultat
Primăvară	Zi	-	Zi de primăvară senină
Primăvară	Zi	Ploaie	Zi de primăvară ploioasă
Primăvară	Zi	Ninsoare	-
Primăvară	Zi	Ceață	Zi de primăvară încețoșată
Primăvară	Noapte	-	Noapte de primăvară senină
Primăvară	Noapte	Ploaie	Noapte de primăvară ploioasă
Primăvară	Noapte	Ninsoare	-
Primăvară	Noapte	Ceață	Noapte de primăvară încețoșată
Vară	Zi	-	Zi de vară senină
Vară	Zi	Ploaie	Zi de vară ploioasă
Vară	Zi	Ninsoare	-
Vară	Zi	Ceață	Zi de vară încețoșată
Vară	Noapte	-	Noapte de vară senină
Vară	Noapte	Ploaie	Noapte de vară ploioasă
Vară	Noapte	Ninsoare	-
Vară	Noapte	Ceață	Noapte de Vară încețoșată
Toamnă	Zi	-	Zi de toamnă senină
Toamnă	Zi	Ploaie	Zi de toamnă ploioasă
Toamnă	Zi	Ninsoare	-
Toamnă	Zi	Ceață	Zi de toamnă încețoșată
Toamnă	Noapte	-	Noapte de toamnă senină
Toamnă	Noapte	Ploaie	Noapte de toamnă ploioasă
Toamnă	Noapte	Ninsoare	-
Toamnă	Noapte	Ceață	Noapte de toamnă încețoșată
Iarnă	Zi	-	Zi de iarnă senină
Iarnă	Zi	Ploaie	-
Iarnă	Zi	Ninsoare	Zi de iarnă cu ninsoare
Iarnă	Zi	Ceață	Zi de iarnă încețoșată
Iarnă	Noapte	-	Noapte de iarnă senină
Iarnă	Noapte	Ploaie	-
Iarnă	Noapte	Ninsoare	Noapte de iarnă cu ninsoare
Iarnă	Noapte	Ceață	Noapte de iarnă încețoșată

Tabel 5.8 – Tabelul combinațiilor de anotimpuri, momente ale zilei și fenomene meteorologice

5.5.4. Sistemul de îmbunătățiri al atributelor

Sistemul de îmbunătățire al atributelor este esențial în orice tip de joc, deoarece reprezintă progresul jucătorului și contribuie la creșterea acestuia prin posibilitatea de evoluare a personajului. Îmbunătățind atributelor precum: viață, rezistență sau forță fizică.



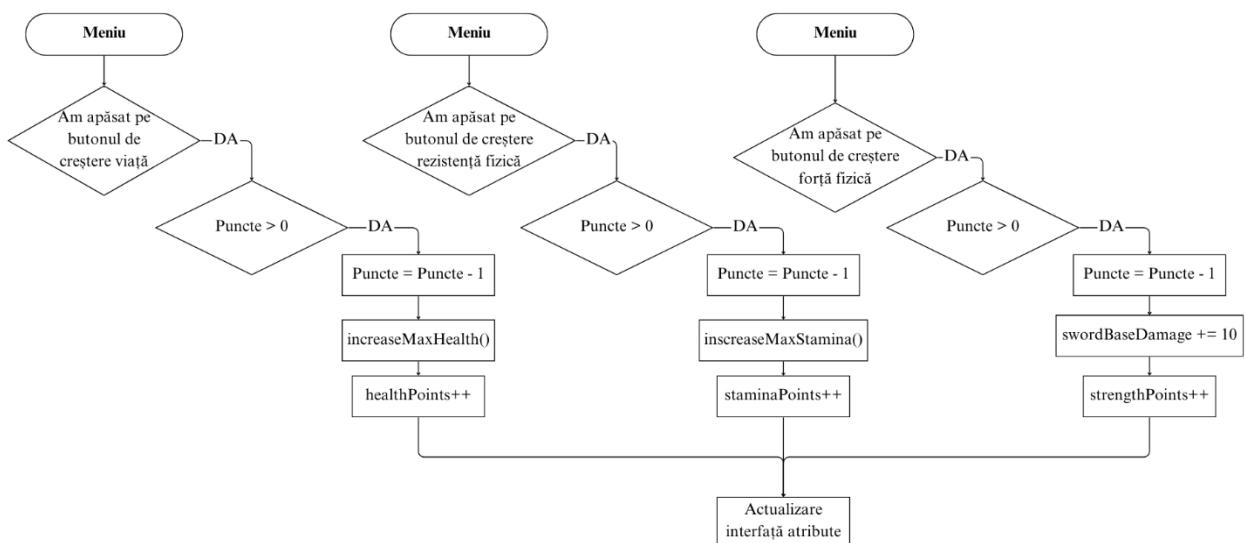
Figură 5.123 – Butonele pentru îmbunătățire atribute și resetare puncte

Secțiunea Level Up afișează jucătorului stadiul de progres actual al personajului, cum ar fi nivelul, experiența acumulată și punctele de nivel curente.

Secțiunea Attribute Points permite vizualizarea punctelor de nivel alocate fiecărui atribut în parte. Prin apăsarea butonului „+”, jucătorul poate îmbunătăți attributele personajului, iar butonul „Reset Points” va reseta toate punctele atribuite pentru fi realocate în funcție de preferințe.

Îmbunătățirea de atribute

Organograma generală a îmbunătățirii de atribute

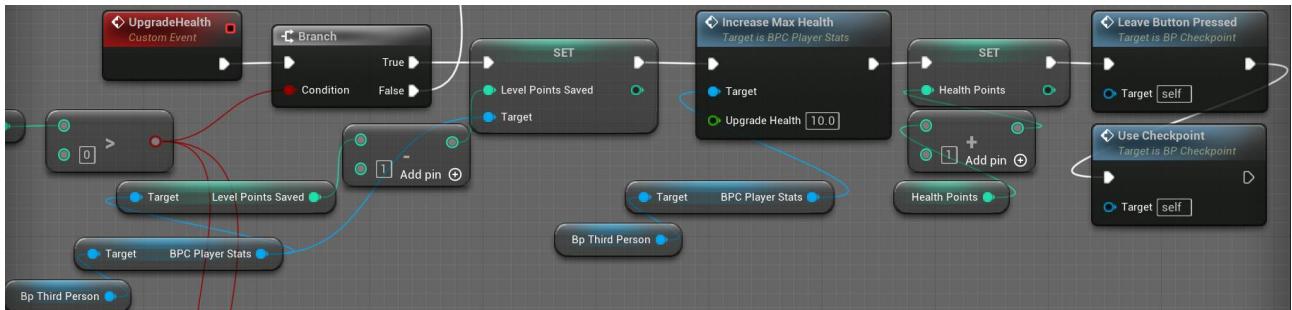


Figură 5.124 – Organograma generală a îmbunătățirii de atribute

Funcțiile de îmbunătățire utilizate în cadrul acestui sistem sunt: **upgradeHealth()**, **upgradeStamina()** și **upgradeStrength()**.

- **upgradeHealth()** – funcție folosită pentru creșterea de viață a personajului.
- **upgradeStamina()** – funcție folosită pentru creșterea de rezistență a personajului.
- **upgradeStrength()** – funcție folosită pentru creșterea de putere a personajului.

Ca exemplu o să prezentăm doar funcția upgradeHealth() deoarece aceasta logică se aplică la fel pentru celelalte două funcții de rezistență și forță fizică. [Figură 5.125]



Figură 5.125 – Funcția de îmbunătățire a vieții (Blueprint)

```

Prin interacțiunea cu butonul de viață apelăm funcția upgradeHealth()
IF(levelPointsSaved > 0), THEN
    | levelPointsSaved = levelPointsSaved - 1
    | Incrementăm viață maximă, increaseMaxHealth(10)
    | healPoints = healPoints + 1
    | leaveButtonPressed()
    | useCheckpoint()
END IF

Prin interacțiunea cu butonul de rezistență apelăm funcția upgradeStamina()
IF(levelPointsSaved > 0), THEN
    | levelPointsSaved = levelPointsSaved - 1
    | Incrementăm rezistență maximă, increaseMaxStamina(10)
    | staminaPoints = staminaPoints + 1
    | leaveButtonPressed()
    | useCheckpoint()
END IF

Prin interacțiunea cu butonul de putere apelăm funcția upgradeStrength()
IF(levelPointsSaved > 0), THEN
    | levelPointsSaved = levelPointsSaved - 1
    | Incrementăm puterea, swordBaseDamage = swordBaseDamage + 10
    | strengthPoints = strengthPoints + 1
    | leaveButtonPressed()
    | useCheckpoint()
END IF

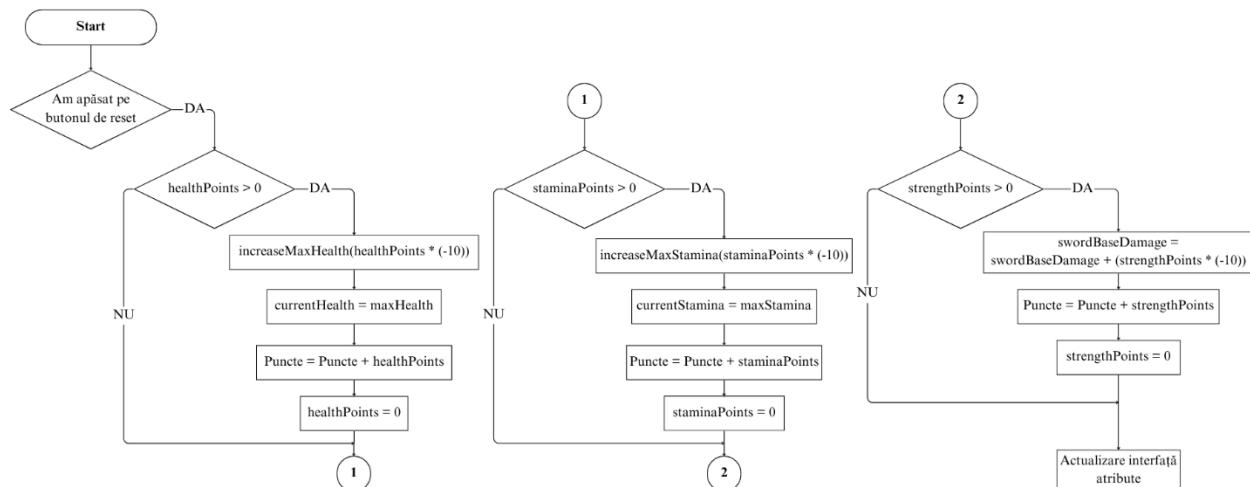
```

- **levelPointsSaved** – variabilă de tip integer, responsabilă de contorizarea punctelor de nivel pentru îmbunătățirea atributelor.
- **healPoints** – variabilă de tip integer, responsabilă de contorizarea punctelor atribuite în atribut viață.
- **staminaPoints** – variabilă de tip integer, responsabilă de contorizarea punctelor atribuite în atributul de rezistență fizică.
- **strengthPoints** – variabilă de tip integer, responsabilă de contorizarea punctelor atribuite în atributul de putere fizică.

- **swordBaseDamage** – varibilă de tip float, responsabilă de creșterea daunelor de armă.
- **leaveButtonPressed()** și **useCheckpoint()** – utilizate pentru a reseta și actualiza UI-ul meniul.

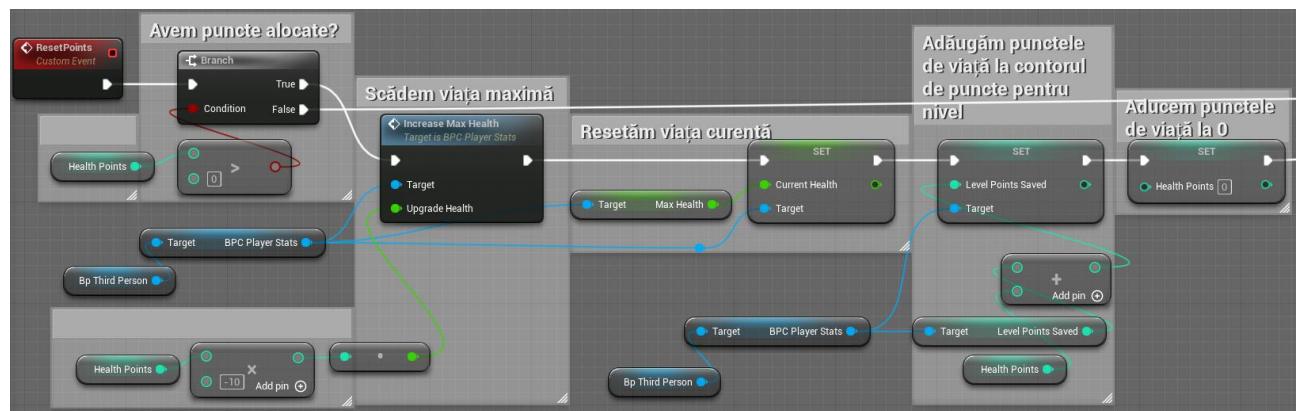
Resetarea de attribute

Organograma generală a resetării de attribute



Figură 5.126 – Organograma generală a resetării de attribute

O să folosim ca exemplu logica de resetare pentru viață deoarece se aplică și pentru celelalte attribute, permitând astfel resetarea punctelor de nivel și restaurarea atributelor default. [Figură 5.127]



Figură 5.127 – Funcția de resetare attribute (Blueprint)

Pentru a scădea attributele, o să folosim **tot funcțiile de increase...** doar că vom înmulții numărul de puncte alocate ale unui atribut $\ast (-10)$ pentru a resetat la valoare inițială de la început.

```

Prin interacțiunea cu butonul de reset apelăm funcția resetPoints()

IF(healPoints > 0), THEN
|   increaseMaxHealth(healPoints * (-10))
|   currentHealth = maxHealth
|   levelPointsSaved = levelPointsSaved + healPoints
|   healPoints = 0
END IF

IF(staminaPoints > 0), THEN
|   increaseMaxStamina(staminaPoints * (-10))
|   currentStamina = maxStamina
|   levelPointsSaved = levelPointsSaved + staminaPoints
|   staminaPoints = 0
END IF

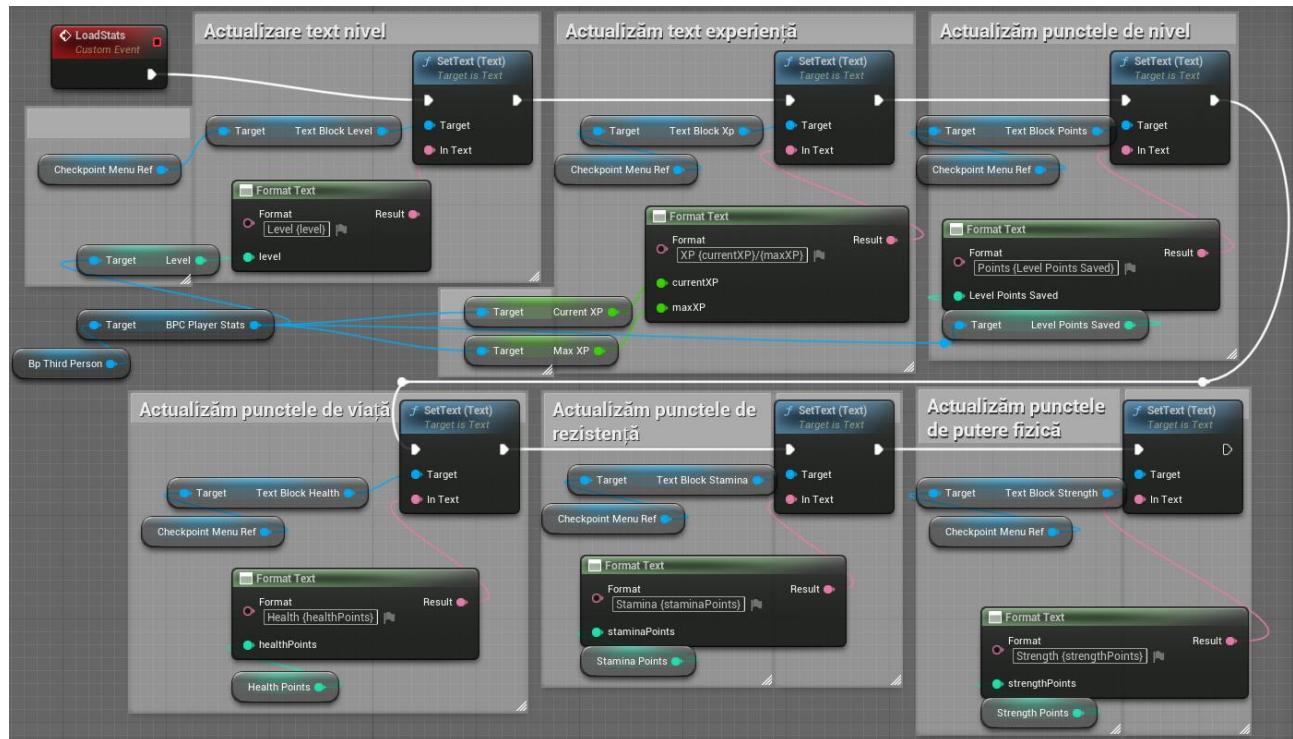
IF(strengthPoints > 0), THEN
|   swordBaseDamage = swordBaseDamage + (strengthPoints * (-10))
|   levelPointsSaved = levelPointsSaved + strengthPoints
|   strengthPoints = 0
END IF

leaveButtonPressed() //reset
useCheckpoint() //reset

```

Actualizarea statisticilor

Actualizarea statisticelor se va face în permanență la începutul interacțiunii cu punctul de acces dar și după fiecare apăsare a butoanelor de îmbunătățire atribute.



Figură 5.128 – Funcția de actualizare statistici (Blueprint)

Este important ca attributele și punctele să fie actualizate în timp real, pentru ca jucătorul să vadă imediat impactul deciziilor sale. Astfel se asigură o experiență interactivă și intuitivă, oferind feedback vizual clar pentru fiecare modificare făcută.

Prin interacțiunea cu checkpoint-ul apelăm funcția loadStats()

Actualizăm textul de nivel din meniu
Actualizăm textul de experiență din meniu
Actualizăm textul punctelor de nivel din meniu
Actualizăm textul punctelor de viață din meniu
Actualizăm textul punctelor de rezistență din meniu
Actualizăm textul punctelor de putere din meniu

5.5.5. Sistemul de schimbare al înfățișării

Sistemul permite jucătorului posibilitatea de a schimba echipamentul vizual prin piesele de armură și îmbrăcăminte pe care acesta le deține.



Figură 5.129 – Slidere RGB și preview de culoare

Sistemul de customizare este alcătuit din cinci piese de armură esențiale și anume:

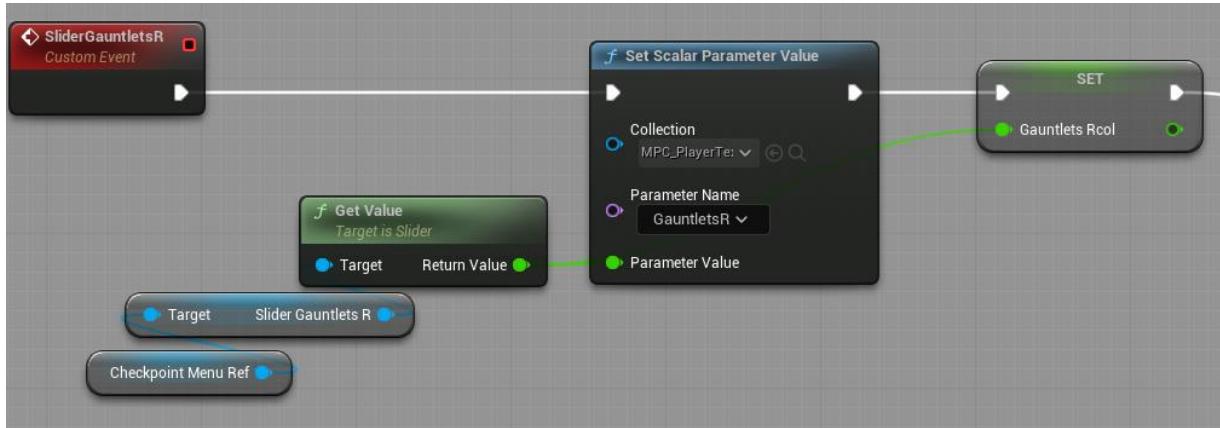
- **Gauntlets** – mănuși
- **Chestplate** – armură de corp
- **Greaves** – protecții pentru picioare
- **Shirt** - cămașă
- **Trousers** – pantaloni

Fiecare cu trei slideri R, G, B pentru a determina culoarea fiecărei piese de armură individuale.

Fiecare componentă RGB a pieselor de armură vor fi stocate într-un **Material Parameter** pentru a fi controlate mai ușor setate default cu valori de început.

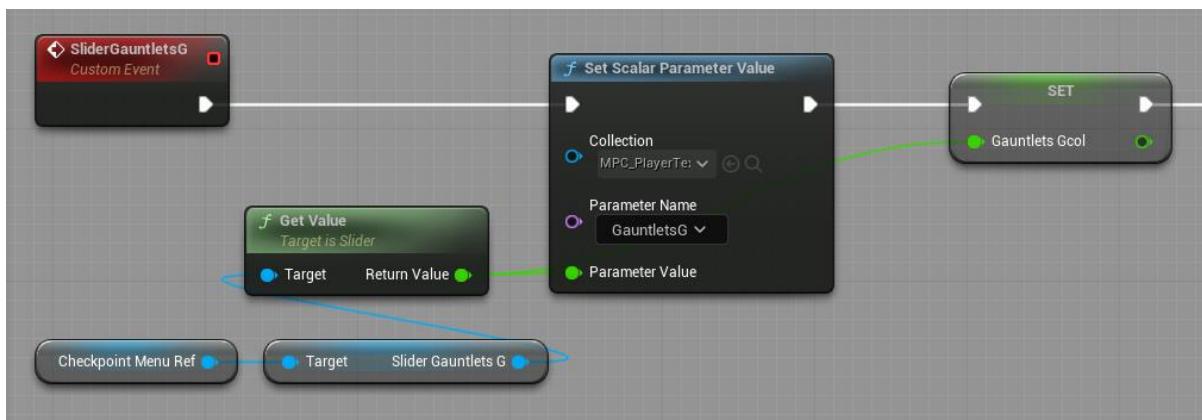
Întrucât funcțiile vor fi identice, o să detaliem doar un exemplu. Aceasta fiind logica în care se poate realiza culoarea RGB a unei piese de armură, în exemplu, cea a mănușilor.

Funcția **sliderGauntletsR()** prin slider-ul acestuia poate lua valori între [0:1], valoare care va fi salvată în variabila de tip float **GauntletsRcol** responsabilă cu stocarea culorii roșu. [Figură 5.130]



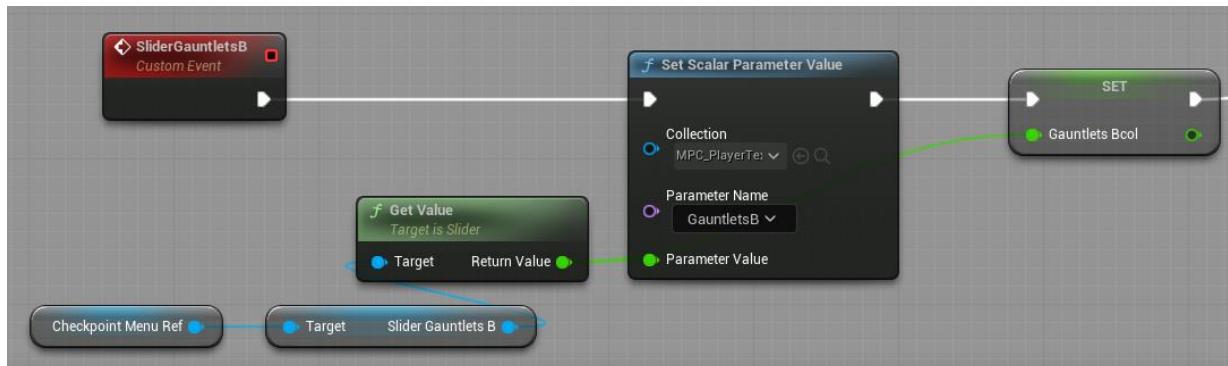
Figură 5.130 – Slidere pentru setarea componentei Red (Blueprint)

Funcția **sliderGauntletsG()** prin slider-ul acestuia poate lua valori între [0:1], valoare care va fi salvată în variabila de tip float **GauntletsGcol** responsabilă cu stocarea culorii verde. [Figură 5.131]



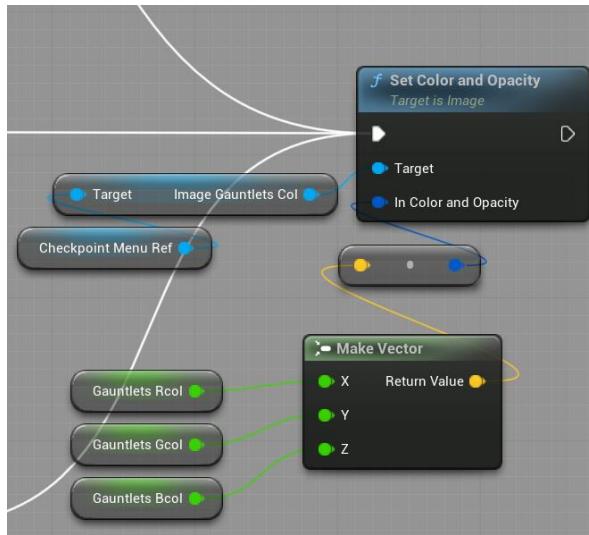
Figură 5.131 – Slidere pentru setarea componentei Green (Blueprint)

Funcția **sliderGauntletsB()** prin slider-ul acestuia poate lua valori între [0:1], valoare care va fi salvată în variabila de tip float **GauntletsBcol** responsabilă cu stocarea culorii albastru. [Figură 5.132]



Figură 5.132 – Slidere pentru setarea componentei Blue (Blueprint)

Pentru a determina culoare RGB a unei piese de armură, preluăm fiecare componentă de culoare (roșu, verde, albastru) și le vom salva într-un vector pentru a determina combinația de culoare pentru imaginea din meniu. [Figură 5.133]

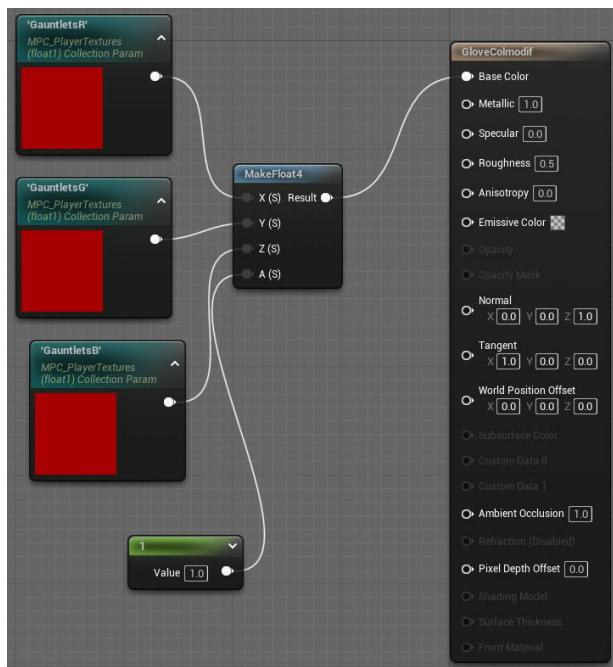


Figură 5.133 – Slidere pentru setarea componentei Blue (Blueprint)

Valorile salvate în Material parameter vor avea rol în a schimba **culoare texturii** a fiecărui material de piesă a armurii. [Figură 5.134]



Figură 5.134 – Setarea parametrilor RGB în MPC



Figură 5.135 – Slidere pentru setarea componentei Blue

Reprezentarea vizuală a schimburi de înfăţişare



Figură 5.136 – Reprezentarea vizuală a schimbului de înfăţişare

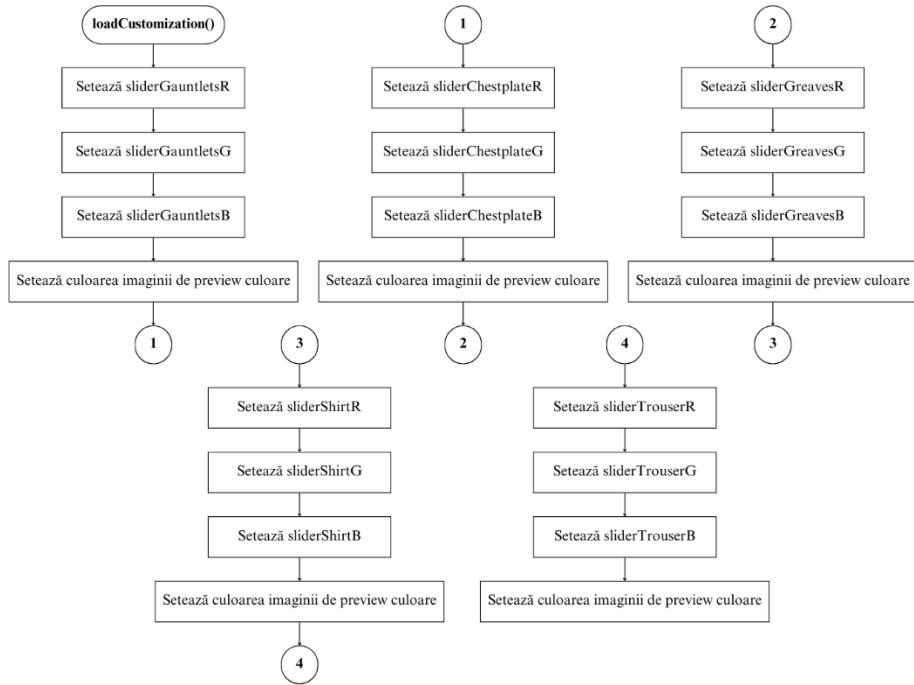
Actualizarea de înfăţişare

Actualizarea de înfăţişare este realizată prin **interacțiunea cu punctul de acces și la fiecare modificare a componentelor RGB ale pieselor de armură**, printr-un manager de customizare care încarcă variabilele cu proprietăatile RGB ale părților de armură pentru a seta sliderele și imaginile preview din meniul de customizare al jucătorului.



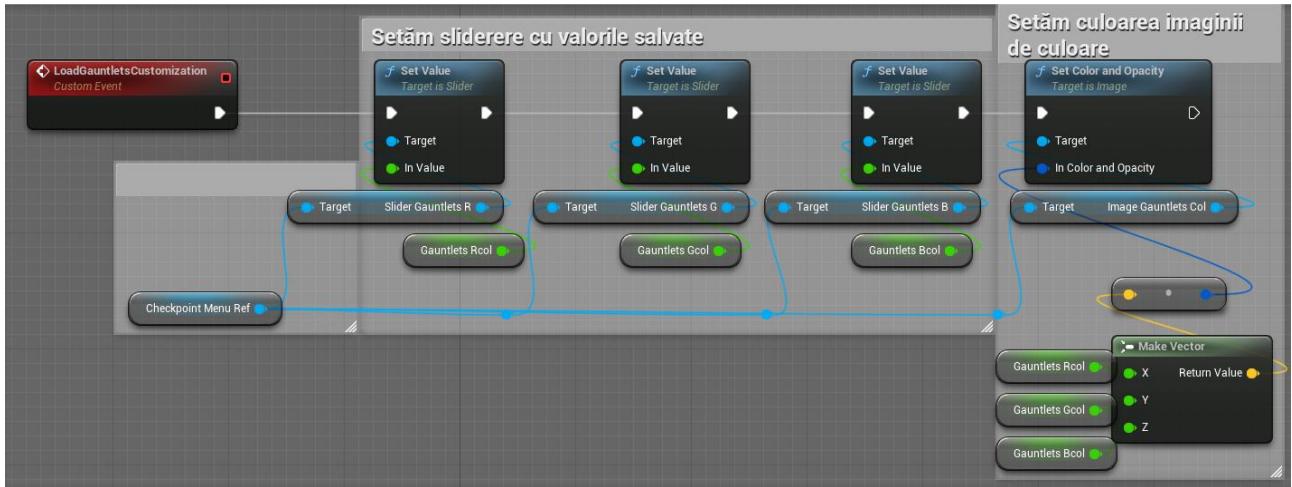
Figură 5.137 – Actualizarea de înfăţişare (Blueprint)

Organograma generlă a managerului de înfățișare



Figură 5.138 – Organograma generală de actualizarea de înfățișare

Aceasta fiind logica prin care se actualizează sliderele de culoare RGB. Întrucât funcțiile vor fi identice, o să detaliem doar un exemplu, cea a mănușilor. [Figură 5.139]



Figură 5.139 – Actualizarea de înfățișare (Blueprint)

În acest mod am reprezentat întregul sistemul punct de acces (Checkpoint), cu cele trei mari sisteme ale acestuia: **Sistemul de vreme**, **Sistemul de îmbunătățiri atribută** și **Sistemul de schimbare aspect**.

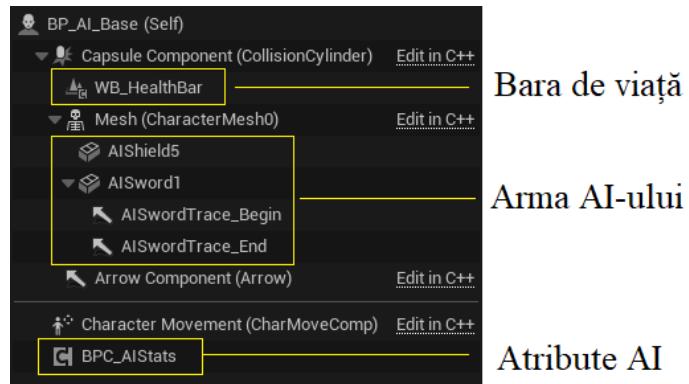
5.6. Sistemul de control AI

Sistemul de control AI reprezintă un ansamblu de mecanisme și componente pentru a simula comportamente ale actorilor non-jucători (NPC) în cadrul jocului. Acesta are rolul de a gestiona logica decizională a oponenților virtuali, permitându-le să reacționeze dinamic în funcție de acțiunile jucătorului și de condițiile de mediu.

În Unreal Engine, acest sistem este alcătuit din componente precum AI Controller, Behaviour Tree, Blackboard și sistemul de percepție, care colaborează pentru a rezulta o logică de tip inteligență artificială.

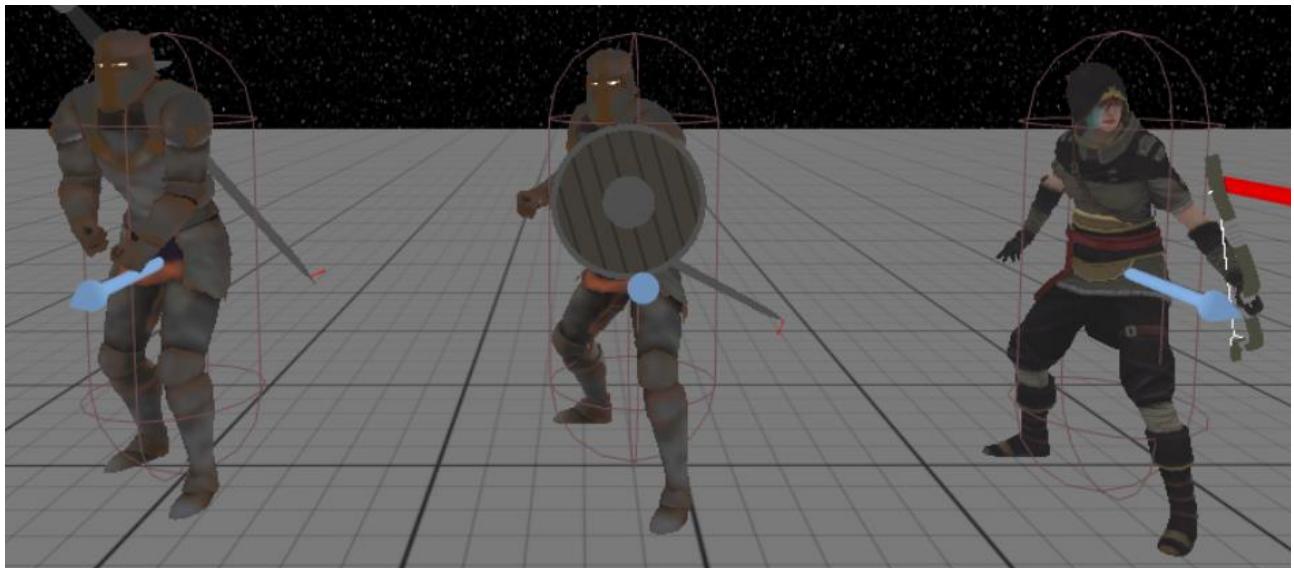
5.6.1. Prezentarea actorului oponent

Actorul oponent, este asemănător actorului personaj controlat de către jucător, cu mici ajustări. Acesta fiind alcătuit din, actorul propriu-zis, bara de viață, arma pe care o deține și atributele acestuia (viață, daune).



Figură 5.140 – Componentele de alcătuire ale actorului oponent

Variante de oponenți



Figură 5.141 – Variante de actori oponent

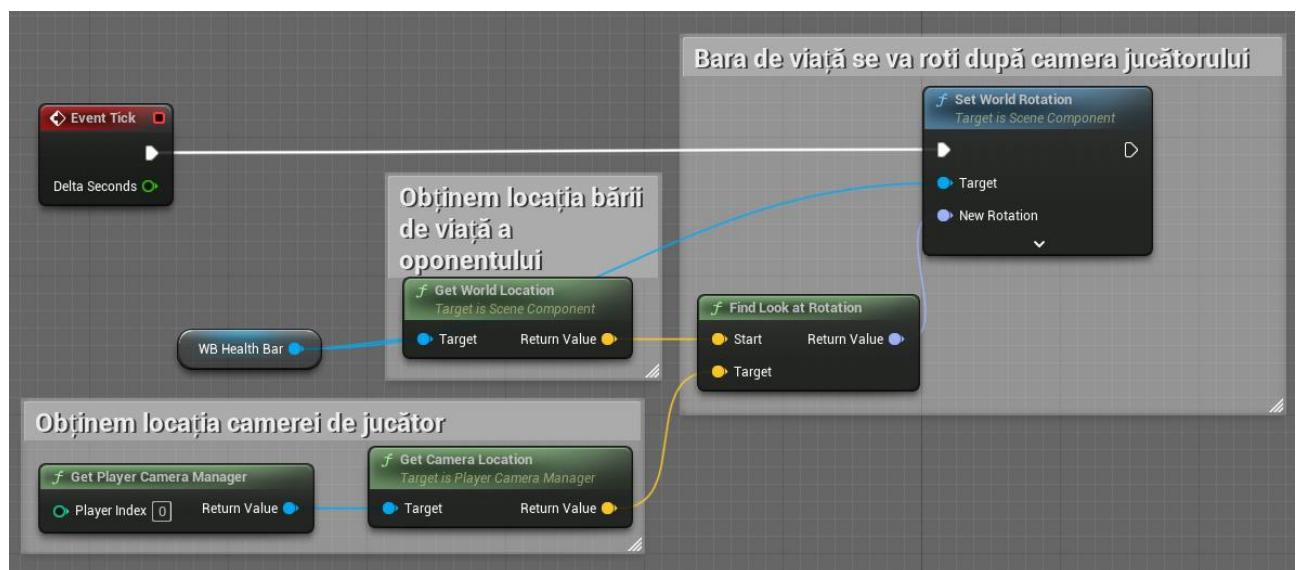
Bara de viață a oponentului

Bara de viață a oponentului reprezintă vizual starea actuală de sănătate a acestuia și nivelul de experiență pe care acesta îl are, oferind jucătorului un indiciu rapid asupra progresului în luptă. [Figură 5.142]



Figură 5.142 – Bara de viață și nivel

Reprezintă un element UI care reflectă dinamic valoare actuală a procentului de viață, actualizându-se în timp real în funcție de daunele primitive. [Figură 5.143]



Figură 5.143 – Rotarea bării de viață și nivel

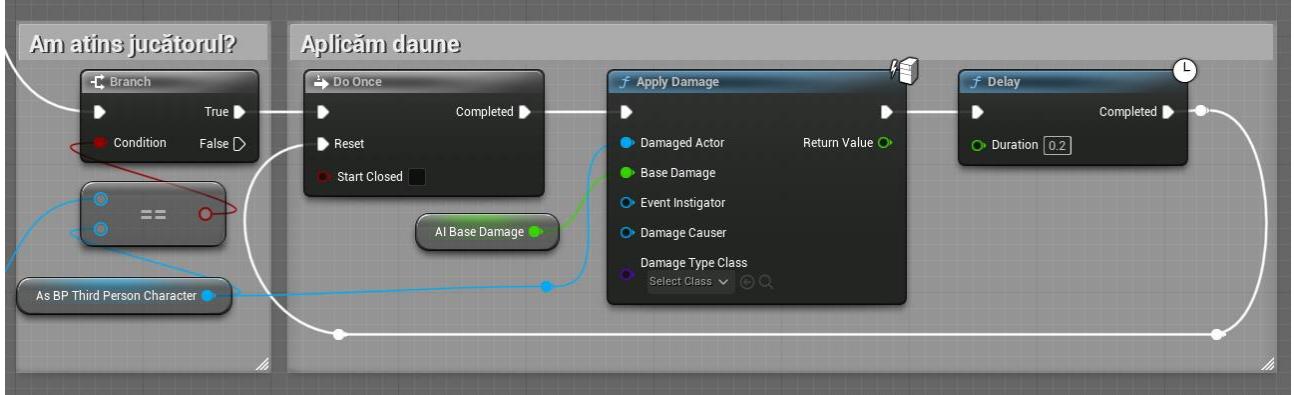
5.6.3. Sub-sistemul de luptă

Sub-sistemul de luptă al oponenților este o versiune simplificată a celui utilizat de jucător, permitând:

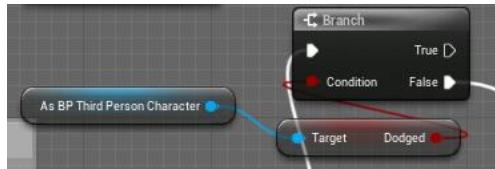
- logica de atac
 - funcționalitatea de echipare și dezechipare a armei
 - trasări de linii pentru atacuri
 - transmiterea daunelor de către oponent asupra jucătorului

Transmiterea daunelor de către oponent mele asupra jucătorului

Transmiterea daunelor către jucător, se realizează similar cu sistemul de aplicare a daunelor folosit de jucător asupra oponenților. Singura diferență constă în faptul că, în cazul atacurilor simultane din partea mai multor oponenți, **daunele sunt aplicate doar de către unul singur, pentru a preveni acumularea excesivă**. [Figură 5.144] Dacă dodge == false jucătorul nu poate primi daune. [Figură 5.145]



Figură 5.144 – Transmiterea daunelor de către oponent mele asupra jucătorului (Blueprint)

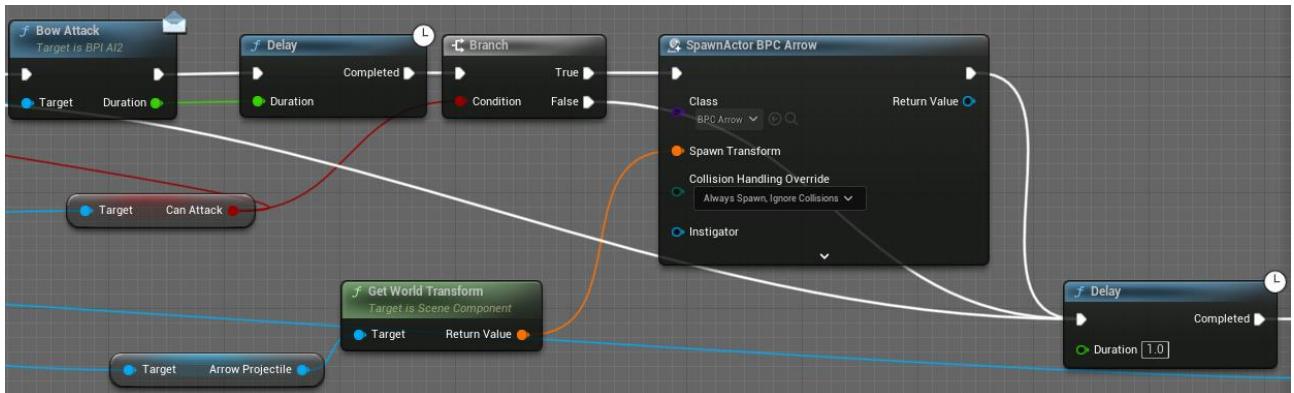


Figură 5.145 – Evitarea de daune (Blueprint)

Transmiterea daunelor de către oponent ranged asupra jucătorului

Diferența dintre mele și ranged este că oponentul mele va fi la o distanță apropiată de jucătorul în momentul provocării daunelor, în timp ce oponentul ranged își va pastra o distanță mai mare.

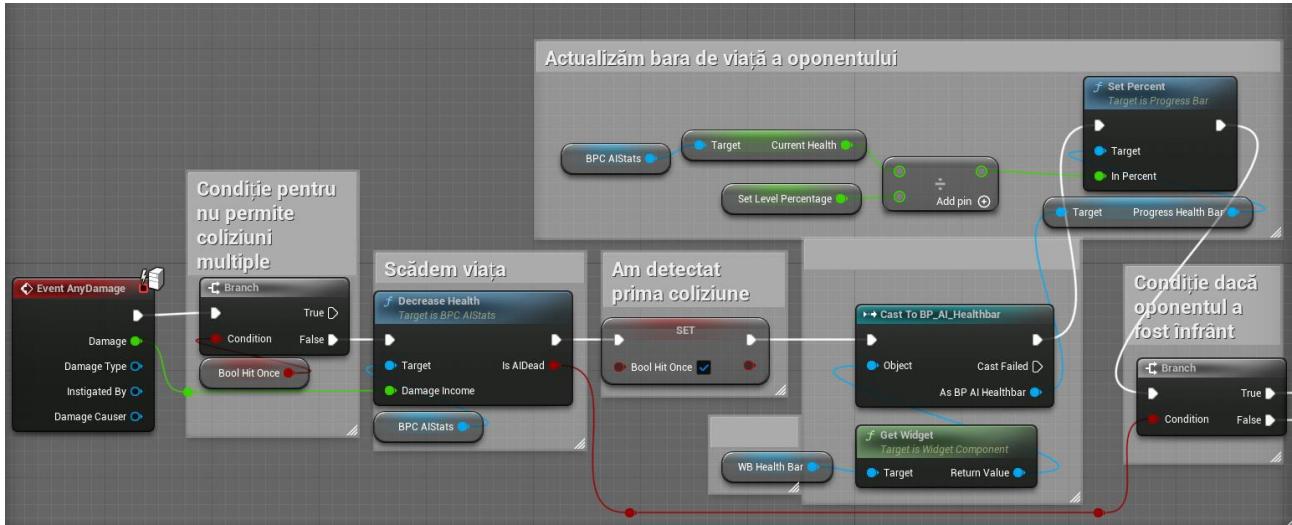
Atacul oponentului ranged este realizat la animația de tragere cu arcul care spawnează în scenă proiectilul săgeată. În momentul coliziunii cu jucătorul va primi daune. De asemenea o să avem grija să ștergem fiecare săgeată din scenă pentru a nu umple scena și consumul de resurse.



Figură 5.146 – Transmiterea daunelor de către oponent ranged asupra jucătorului (Blueprint)

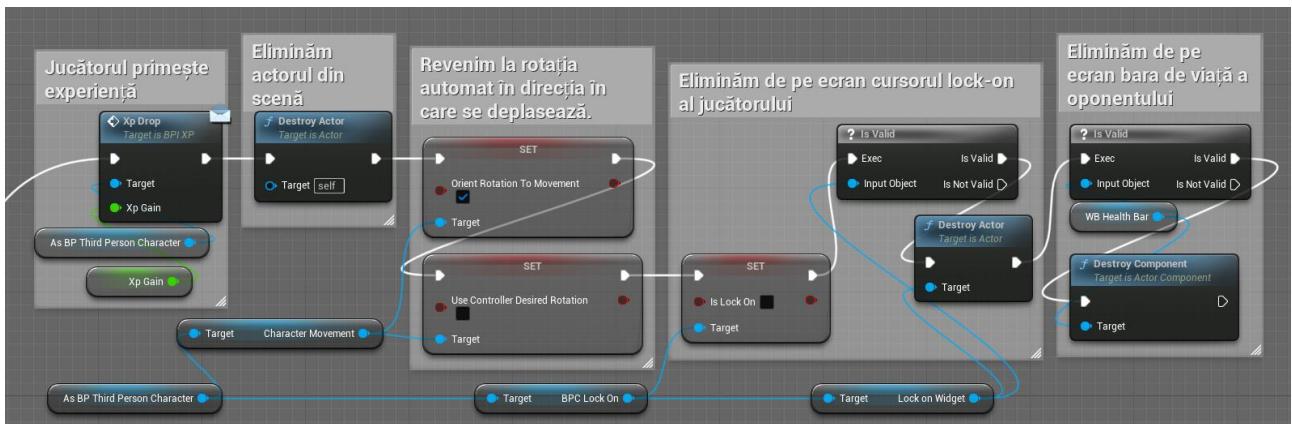
Recepția daunelor de către oponent

În momentul în care oponentul primește daune de la jucător, variabila **boolHitOnce** verifică dacă a fost atins de cel mult o singură coliziune pentru a nu permite daune multiple per atac. Actualizarea bării de viață după lovitură și verificarea condiției de return a funcției **decreaseHealth()**, True dacă oponentul a atins pragul 0 de viață, adică dacă a fost înfrânt. [Figură 5.147]



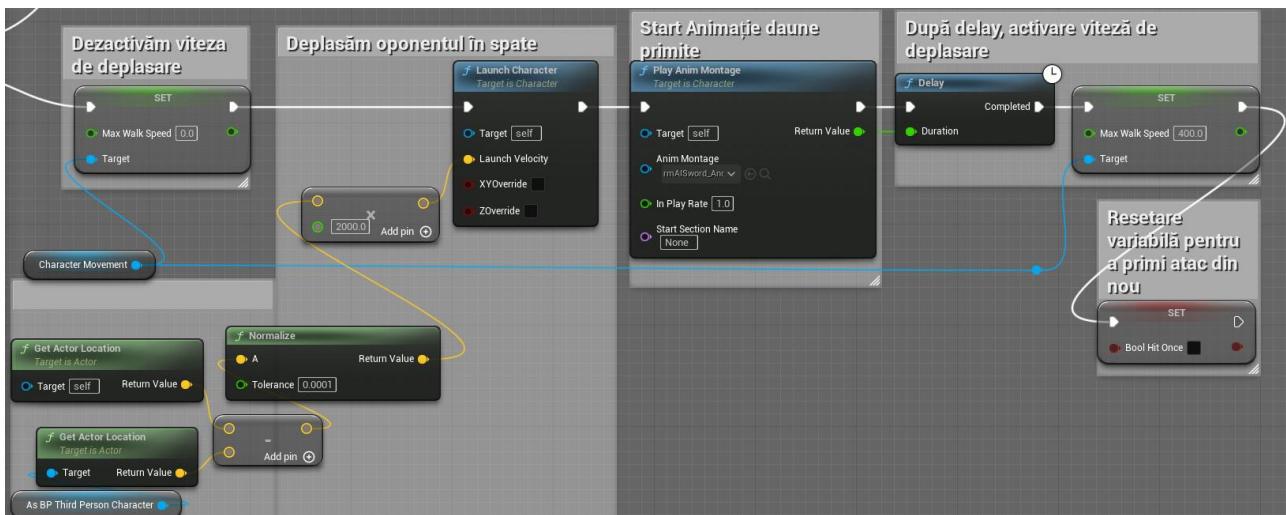
Figură 5.147 – Recepția daunelor de către oponent (Blueprint) (1)

Condiția de True: Jucătorul va primi o cantitate de experiență drept recompensă. De asemenea eliminarea din scenă a componentelor precum, actorul învins, cursorul de fixare (lock-on) și bara de viață, pentru a evita păstrarea în scenă a unor elemente, care nu mai au contribuție în joc. [Figură 5.148]



Figură 5.148 – Recepția daunelor de către oponent (Blueprint) (2)

Condiția de False: Oponentul nu a atins încă pragul de 0 viață. În acest caz, viteza de deplasare se va opri pentru un timp în care este împins în spate de către atacul primit, se declanșează animația de lovitură primită, iar după un scurt delay, viteza de deplasare și variabila **boolHitOnce** sunt resetate, permitând astfel jucătorului să aplice din nou daune. [Figură 5.149]



Figură 5.149 – Recepția daunelor de către oponent (Blueprint) (3)

Când oponentul primește daune

```

IF(boolHitOnce == False), THEN
| decreaseHealth()
| boolHitOnce = True
| Actualizăm bara de viață a oponentului
| IF(isAIDead == True), THEN
| | xpDrop()
| | destroyActor()
| | orientRotationToMovement = True
| | useControllerDesiredRotation = False
| | isLockOn = False
| | IF(lockOnWidget == Valid), THEN
| | | destroyActor(widget)
| | | IF(wbHealthBar == Valid), THEN
| | | | destroyComponent()
| | | END IF
| | END IF
| ELSE
| | maxWalkSpeed = 0
| | Deplasăm oponentul în spate
| | Incepe animația de daune
| | delay()
| | maxWalkSpeed = 400
| | boolHitOnce = False
| END IF
END IF

```

- **boolHitOnce** – variabilă de tip boolean, responsabilă cu validarea dacă a fost lovit de cel mult o dată de sferă de coliziune
- **isAIDead** – variabilă de tip boolean care indică învingerea oponentului în momentul când viață atinge pragul 0.
- **isLockOn** – variabilă de tip boolean, reset de control fixat asupra oponentului.
- **maxWalkSpeed** – variabilă de tip float, responsabilă de viteza de deplasare a oponentului.
- **decreaseHealth()** – funcție utilizată pentru a scădea viață.
- **xpDrop()** – funcție de recompensă în momentul în care un oponet este înfrânt.
- **lockOnWidget** – referința cursorului lock-on pe oponent.
- **wbHealthBar** – referința bării de viață a oponentului.

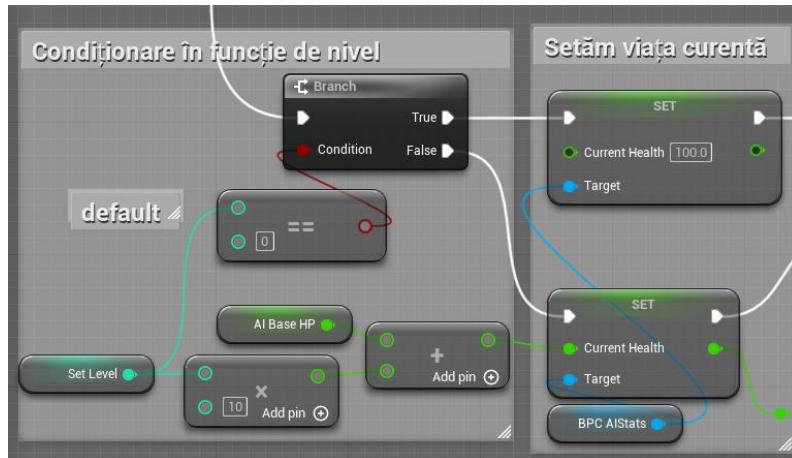
Varietate de nivel a oponenților

Varietate de nivel a oponenților constă în diferențierea caracteristicilor precum viață și daunele provoate. În funcție de nivelul acestuia, vor rezulta:

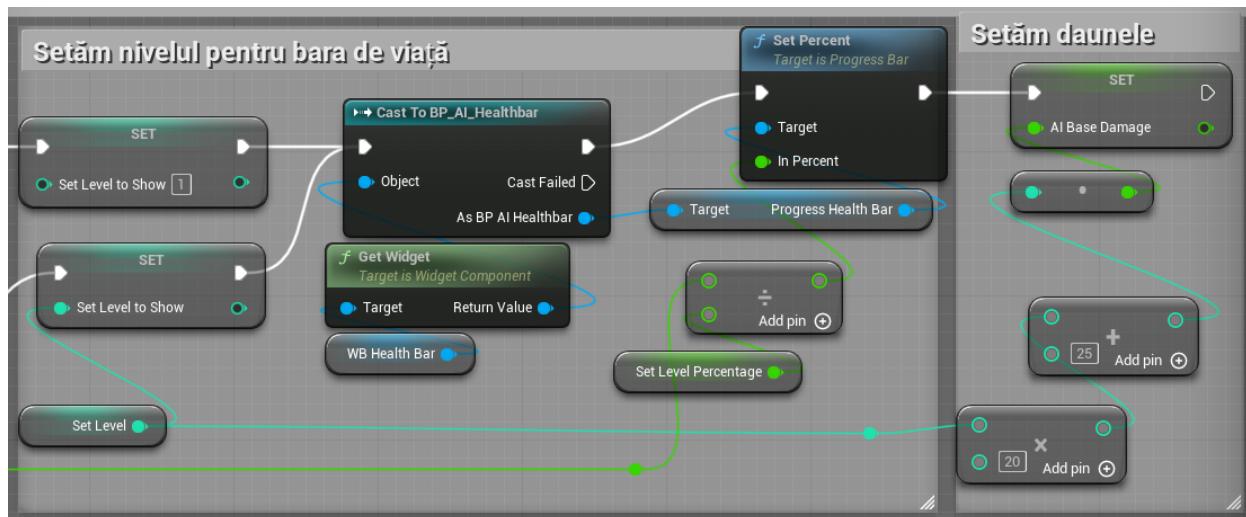
- cantitate de viață, și
- daune de atac sporite

Această varietate contribuie la dificultatea progresivă a jocului și la diversificarea confruntărilor dintre jucător și oponent.

Setarea de viață și daune în funcție de nivel



Figură 5.151 – Setarea de viață și daune în funcție de nivel (Blueprint) (1)



Figură 5.150 – Setarea de viață și daune în funcție de nivel (Blueprint) (2)

- **setLevel** – variabilă de tip integer publică care poate fi modificată în fiecare instanță a oponenților.
- **aiBaseHp** – variabilă de tip float, responsabilă cu setarea de viață default (100)
- **aiBaseDamage** – variabilă de tip float, care indică daunele pe care poate să le provoace.

```

La început de joc
  IF(setLevel = 0), THEN
    | currentHealth = 100
    | setLevelToShow = 1
    | Setăm bara nivelul la bara de viață
    | aiBaseDamage = (setLevel * 20) + 25
  ELSE
    | currentHealth = aiBaseHP + (setLevel * 10)
    | setLevelToShow = setLevel
    | Setăm bara nivelul la bara de viață
    | aiBaseDamage = (setLevel * 20) + 25
END IF

```

Incrementarea graduală a valorilor de viață și daune se vor face prin următoarea formulă:

- **currentHealth** = viață default(100) + (nivel actual * 10)
- **aiBaseDamage** = (nivel actual * 20)

Tabelul de reprezentare a statisticilor oponenților în funcție de nivel

Level	Viață	Daune
1	100 + (1 * 10) = 110	(1 * 20) + 25 = 45
2	100 + (2 * 10) = 120	(2 * 20) + 25 = 70
3	100 + (3 * 10) = 130	(3 * 20) + 25 = 95
4	100 + (4 * 10) = 140	(4 * 20) + 25 = 120
5	100 + (5 * 10) = 150	(5 * 20) + 25 = 145
6	100 + (6 * 10) = 160	(6 * 20) + 25 = 170
7	100 + (7 * 10) = 170	(7 * 20) + 25 = 195
8	100 + (8 * 10) = 180	(8 * 20) + 25 = 220
9	100 + (9 * 10) = 190	(9 * 20) + 25 = 245
10 ...	100 + (10 * 10) = 200	(10 * 20) + 25 = 270

Tabel 5.9 – Tabelul de reprezentare al statisticilor oponenților în funcție de nivel

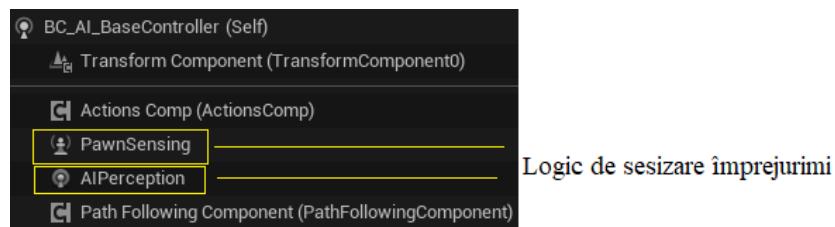
5.6.4. Componentele de control ale sistemului AI

Componentele de control ale sistemul AI includ elemente esențiale precum: **AI Controller**, **Behavior Tree**, **Blackboard** și sistemul de percepti, care lucrează împreună pentru a simula un comportament inteligent al actorilor controlați. Împreună, cele două componente permit dezvoltarea unui AI modular, flexibil și ușor de întreținut.

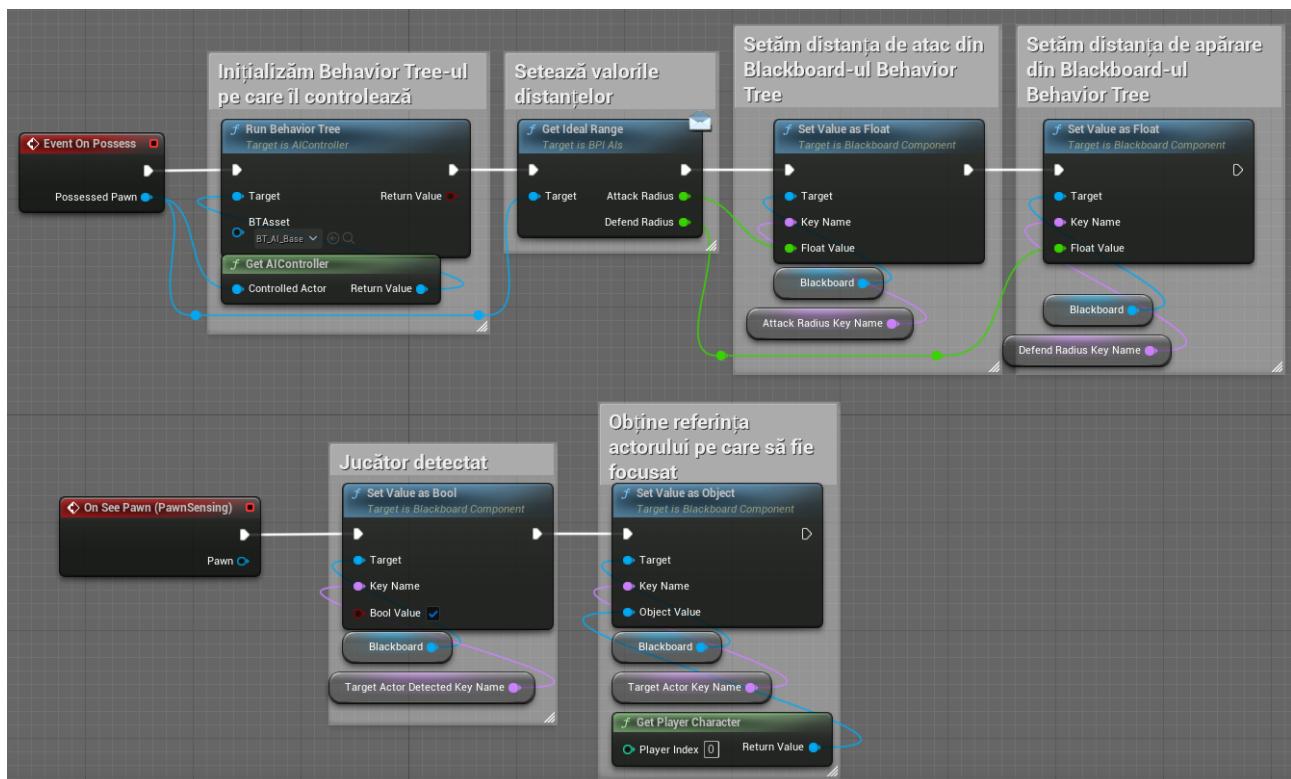
AI Controller

AI controller-ul este componenta creier al unui actor controlat de inteligență artificială. Acesta preia decizii și interacționează cu elemente precum **Behavior Tree**, **Blackboard** și **sistemul de percepție**. Rolul său este de a gestiona comportamentul AI-ului, de a seta focusul, comenzi, mișcare și de interpreta datele colectate din mediu cum ar fi poziția acestuia, calcularea de căi de acces și altele. [32]

Componentele actorului AI controller



Figură 5.152 – Componentele controller-ului AI



Figură 5.153 – Detectarea jucătorului de către AI (Blueprint)

- **EventOnPossess** – se activează la început de joc și initializează Behavior Tree-ul pe care îl controlează.

- **OnSeePawn (PawnSensing)** – este folosit pentru detecția jucătorului dacă acesta este în raza de viziune a AI-ului. Acesta validează starea de detectare activă și referința actorului detectat.

Blackboard & Behaviour Tree

Pentru ca un sistem AI să poată avea acțiuni strucurate și decizii dinamice, Unreal Engine utilizează componente precum Blackboard și Behavior Tree.[\[33\]](#)

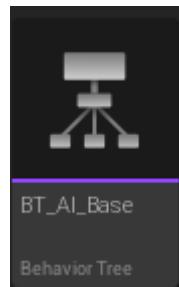
Blackboard, acționează ca o bază de date în care se salvează informații despre AI sub formă de variabile care determină anumite acțiuni în funcție de schimbare acestora. [\[Figură 5.124\]](#)



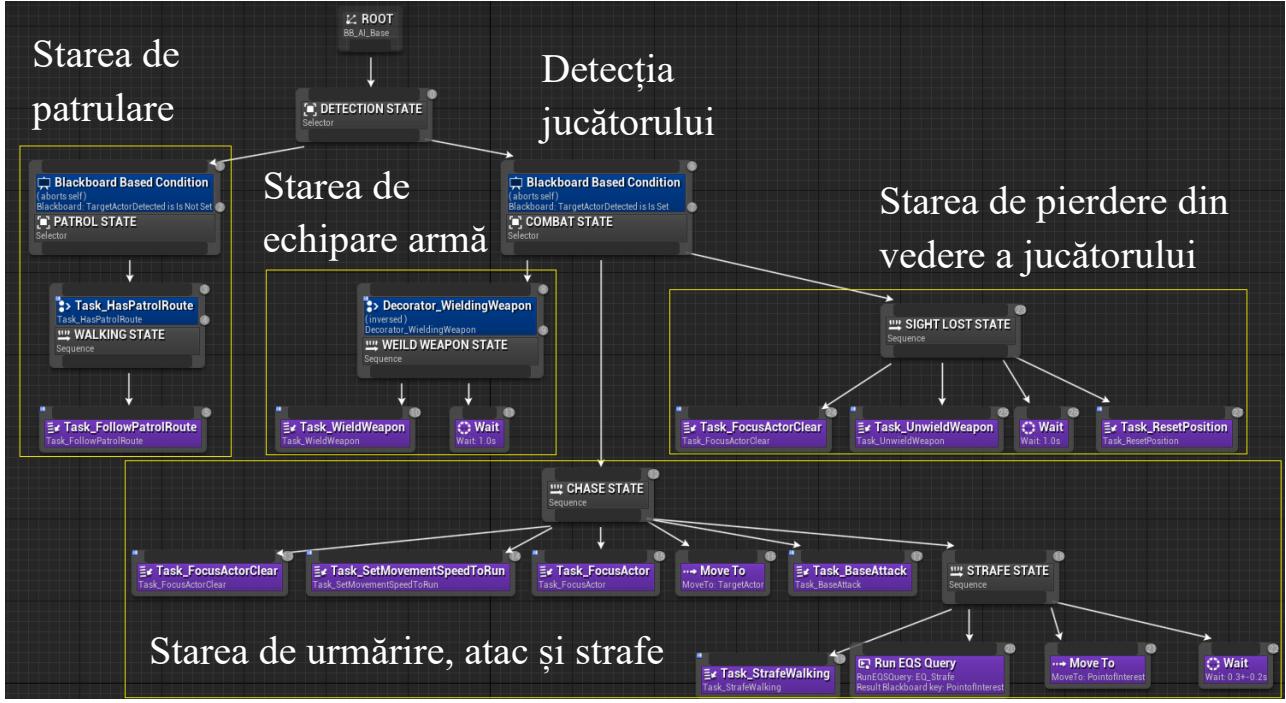
Figură 5.154 – Blackboard-ul AI

- **selfActor** – referință de tip actor, propriu-zis care este controlat.
- **targetActor** – referință de tip actor, pe care AI-ul o focusează (Jucătorul).
- **targetActorDetected** – variabilă de tip boolean, responsabilă cu detectarea jucătorului.
- **pointOfInterest** – variabilă de tip float pentru a determina punctele de rută pentru starea de patrulare.
- **attackRadius** – distanță de la care AI-ul poate ataca jucătorul.
- **defendRadius** – distanță pe care o ia față de jucător pentru a se apăra.

Behavior Tree, reprezintă un arbore decizional care determină logica de comportament a AI-ului în funcție de condițiile din Blackboard. Acesta controlează logica pe care o urmează pentru a îndeplini o anumită acțiune. [\[Figură 5.155\]](#)

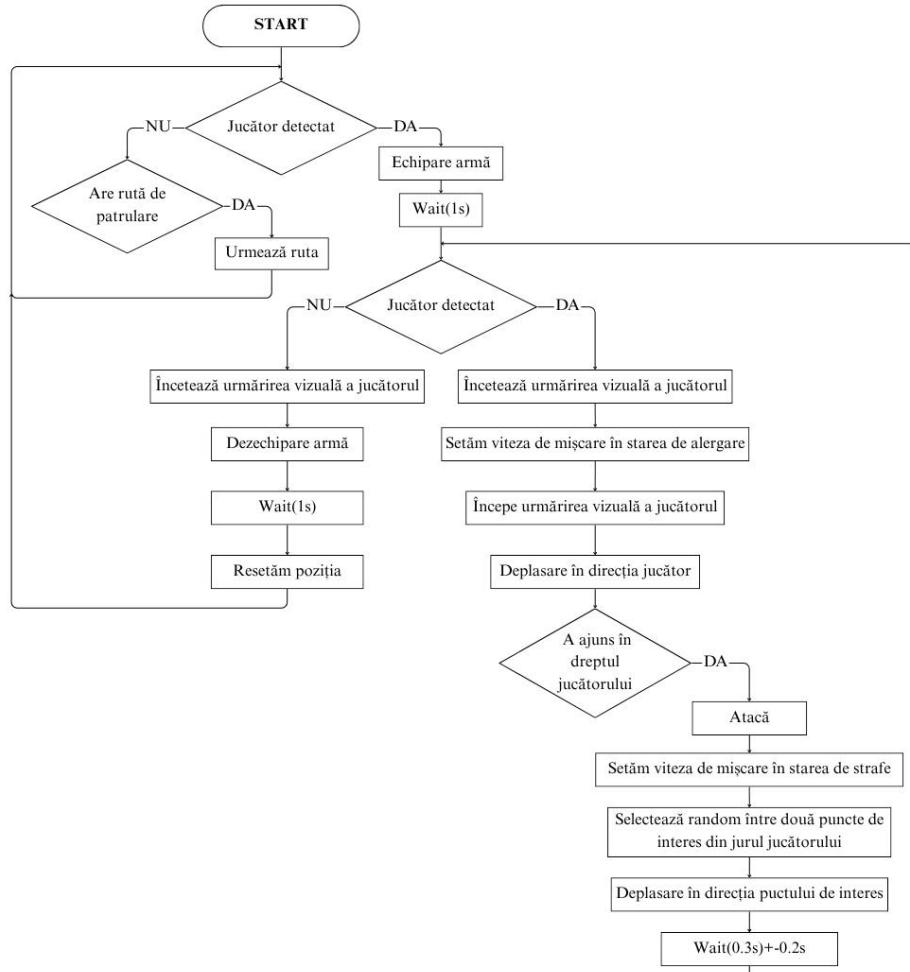


Figură 5.155 – Behavior Tree



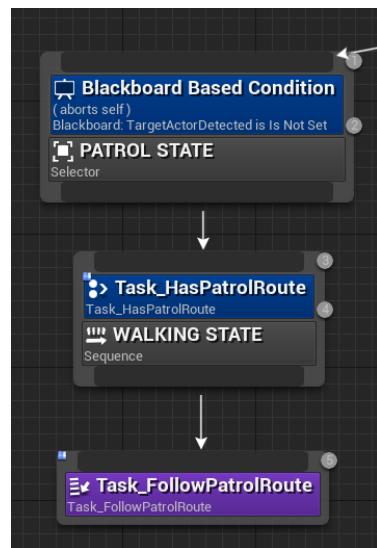
Figură 5.156 – Stările AI (Behavior Tree)

Oranigrama generală a controlului AI



Figură 5.157 – Organigrama generală a controlului AI

Starea de patrulare a AI-ului



Figură 5.158 – Starea de patrulare a AI-ului

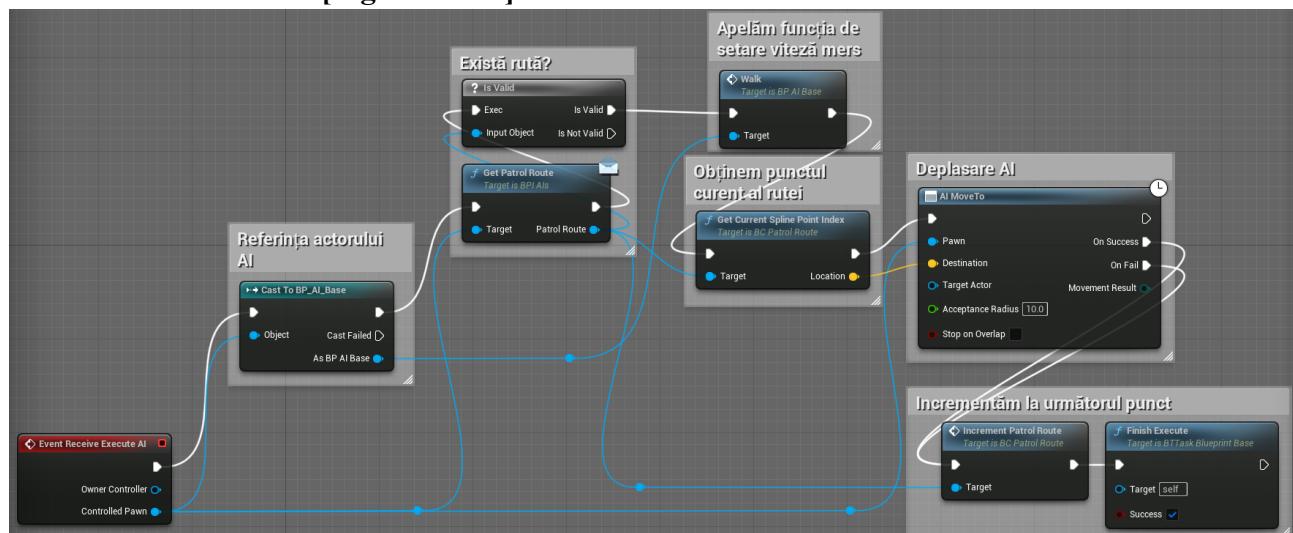
Starea este apelată la începutul jocului și permite AI-ului să se deplaseze în direcția unei rute atât timp cât nu a detectat un jucător în raza acestuia de acțiune.

- **Task_HasPatrolRoute** – este un task personalizat care verifică dacă AI-ul are rută de patrulare. [Figură 5.159]



Figură 5.159 – Task-ul de rută de patrulare

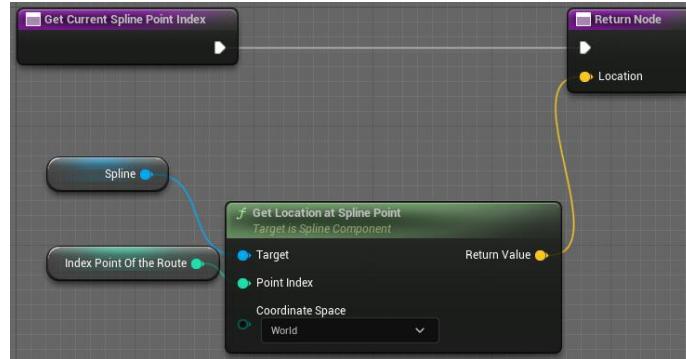
- **Task_FollowPatrolRoute** – este un task personalizat care se ocupă de取得 ruta de deplasare a AI-ului. Aceasta accesează punctele de pe rută și permite deplasare dintr-un punct în altul în buclă. [Figură 5.160]



Figură 5.160 – Task-ul de urmare a rutei

getPatrolRoute() – funcția returnează indexul punctului de pe ruta de patrulare a AI-ului.

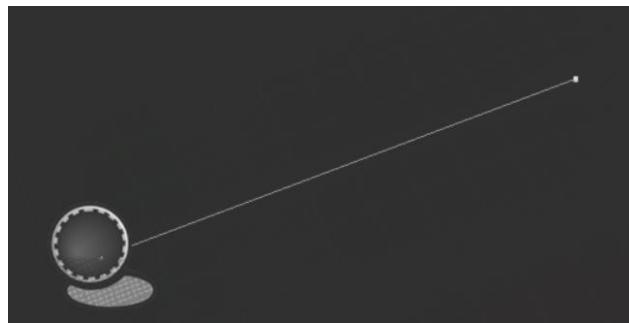
getCurrentSplinePointIndex() – funcția returnează **locația** indexului punct al rutei. [Figură 5.161]



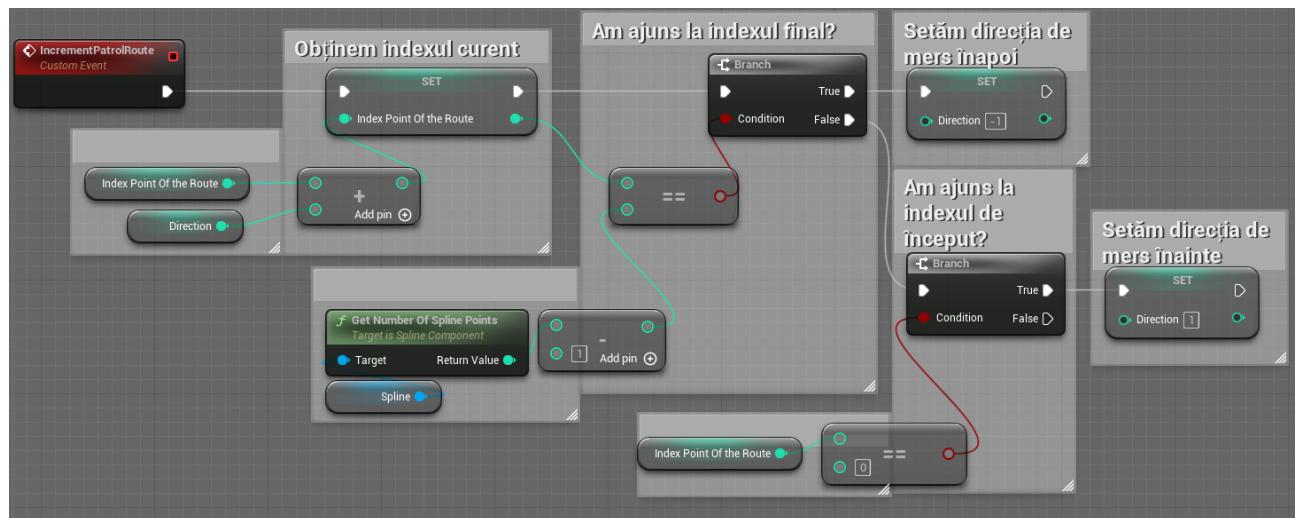
Figură 5.161 – Funcția de obținere a punctelor a liniei de patrulă

Actorul rută

Actorul rută este **o linie de tip spline** care definește traseul pe care AI-ul trebuie să îl urmeze în timpul patrulării. Această linie din exemplu include două puncte, început și sfârșit, dar poate conține și puncte intermediare, permitând astfel mișcări pe direcții variate. Ai-ul va urmări aceste puncte secvențial, simulând astfel un comportament realist de patrulare. [Figură 5.162]



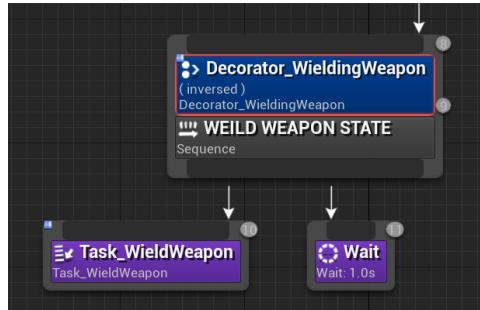
Figură 5.162 – Reprezentarea vizuală a liniei de patrulă



Figură 5.163 – Blueprint a liniei de patrulă

- **indexPointOfTheRoute** – variabilă de tip integer, responsabilă cu reținerea indexului punctelor de rută.
- **direction** – variabila de tip integer, responsabilă în determinarea direcției de deplasare a AI-ului, 1 – în față, -1 – în spate.
- **Nodul getNumberOfSplinePoints** – returnează numărul total de indexi ai rutei.

Starea de echipare armă



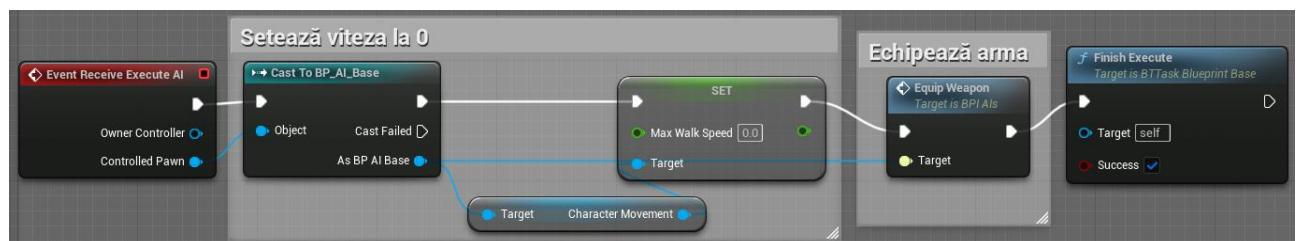
Figură 5.164 – Starea de echipare armă a AI-ului

- **Decorator_WieldingWeapon** – este un decorator personalizat care condiționează execuția nodului următor de stare de echipare a armei AI-ului. [Figură 5.164], [Figură 5.165]

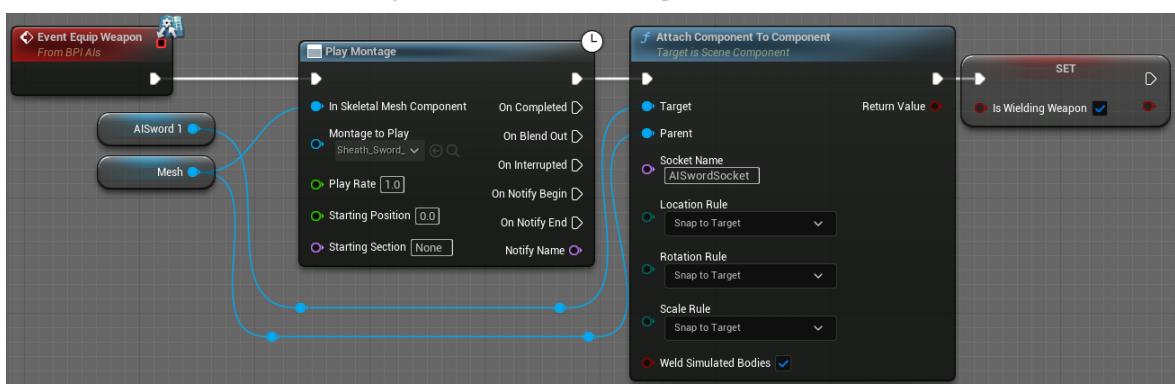


Figură 5.165 – Condiția de echipare a armei AI

- **Task_WieldWeapon** – este un task personalizat care apelează funcția de echipare armă. Funcția `equipWeapon()` este identică cu cea a jucătorului. [Figură 5.166], [Figură 5.167]



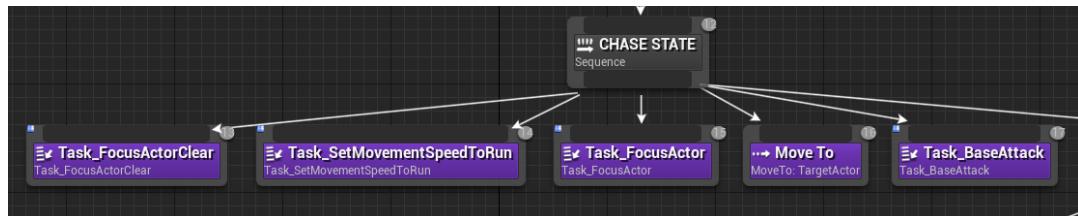
Figură 5.166 – Task-ul de echipare a armei AI



Figură 5.167 – Funcția de echipare armă AI

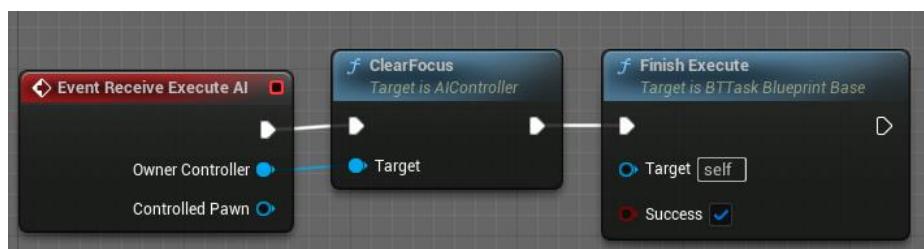
- **Wait** – nod default al sistemului care permite asteptarea pentru un interval de timp.

Starea de urmărire și atac



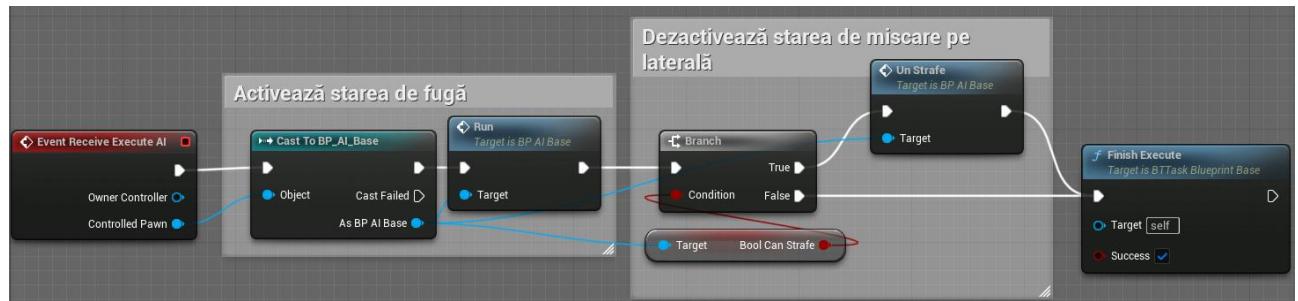
Figură 5.168 – Starea de urmărire și atac a AI-ului

- **Task_FocusActorClear** – este un task personalizat care șterge ținta către care AI-ul își direcționează atenția. [Figură 5.169]



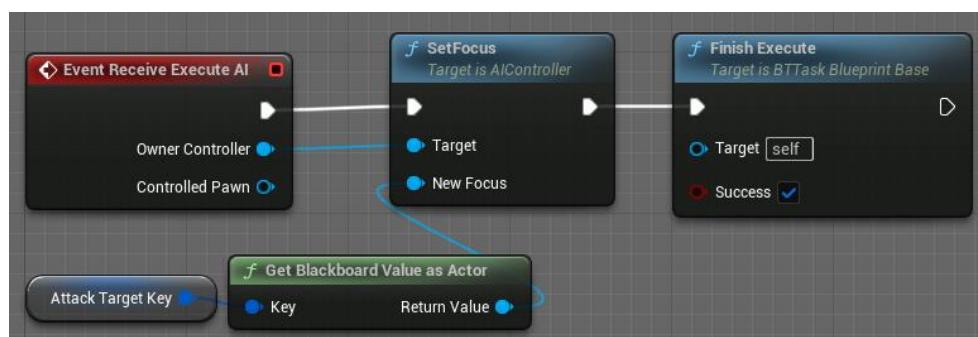
Figură 5.169 – Task-ul de ștergere focusare AI asupra jucătorului

- **Task_SetMovementSpeedToRun** - este un task personalizat care șterge ținta către care AI-ul își direcționează atenția. [Figură 5.170]



Figură 5.170 – Task-ul de deplasare AI spre jucătorului

- **Task_FocusActor** – este un task personalizat care setează direcția de atenție a AI-ului. [Figură 5.171]



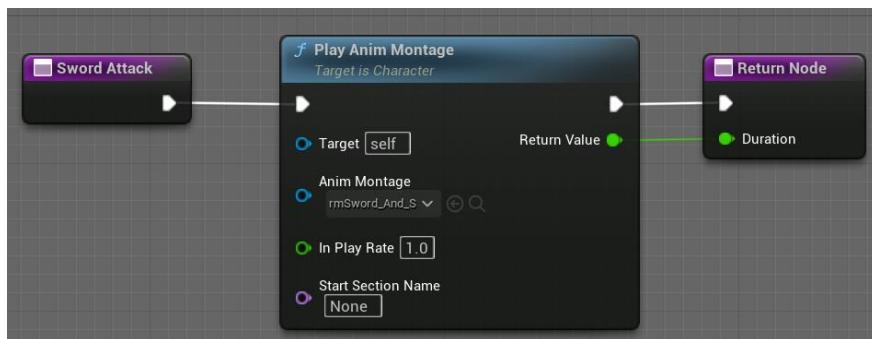
Figură 5.171 – Task-ul de focusare AI asupra jucătorului

- **Move to** – nod default al sistemului care permite deplasarea către o locație a unei referințe de actor.
- **Task_BaseAttack** – este un task personalizat care apelează funcția de atac **swordAttack()**. [Figură 5.172]



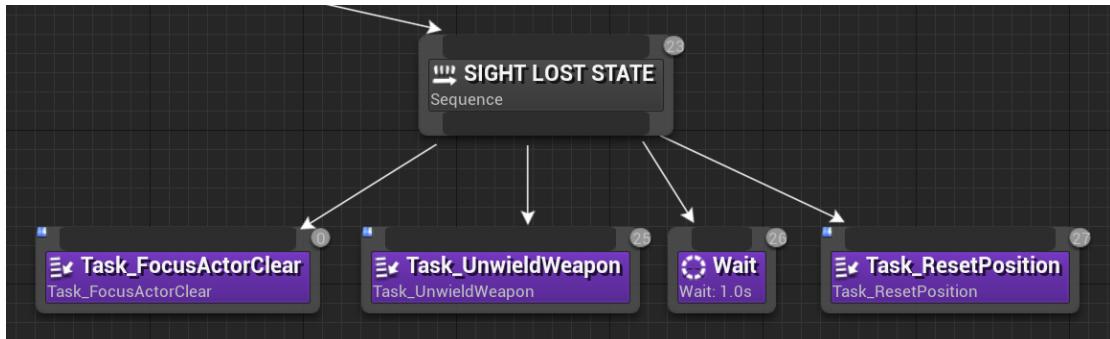
Figură 5.172 – Task-ul de atac AI

Funcția **swordAttack()** – returnează durata animației pentru delay.



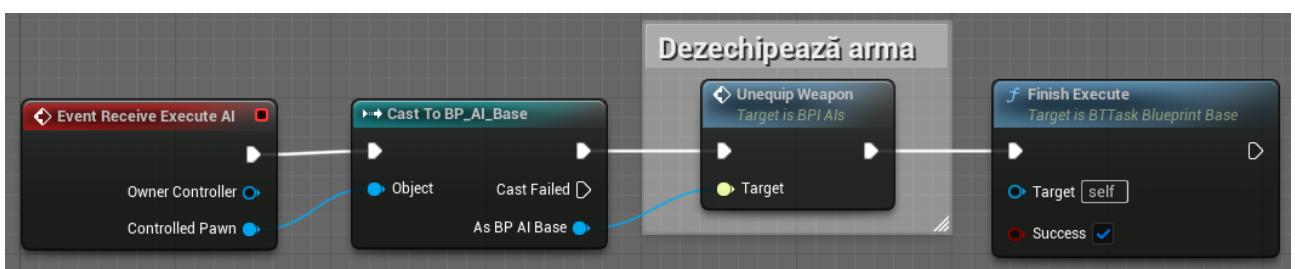
Figură 5.173 – Funcția de atac AI

Starea de pierdere din vedere a jucătorului

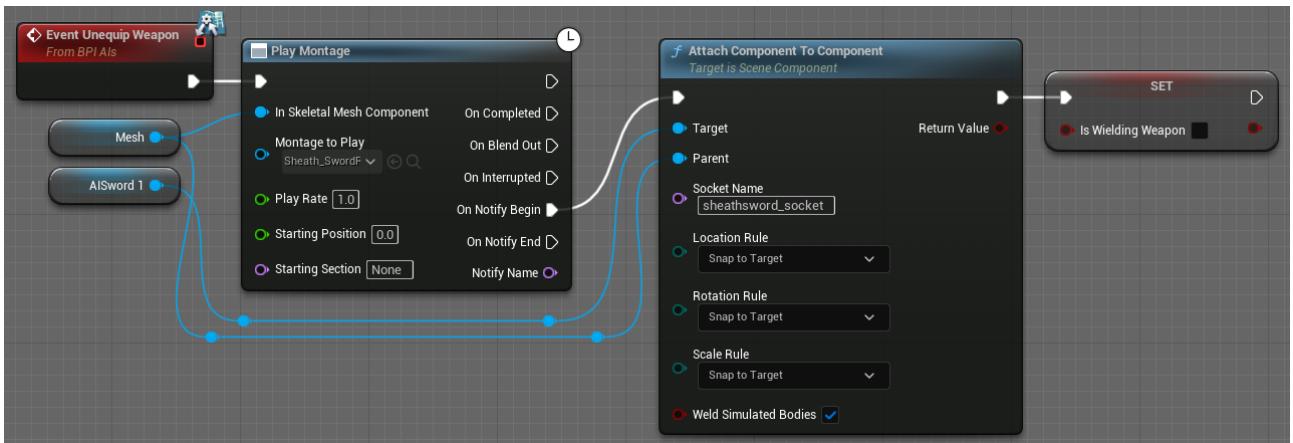


Figură 5.174 – Stare de pierdere din vedere a jucătorului

- **Task_UnwindWeapon** - este un task personalizat care apelează funcția de echipare armă. Funcția **unequipWeapon()** este identică cu cea a jucătorului. [Figură 5.175]

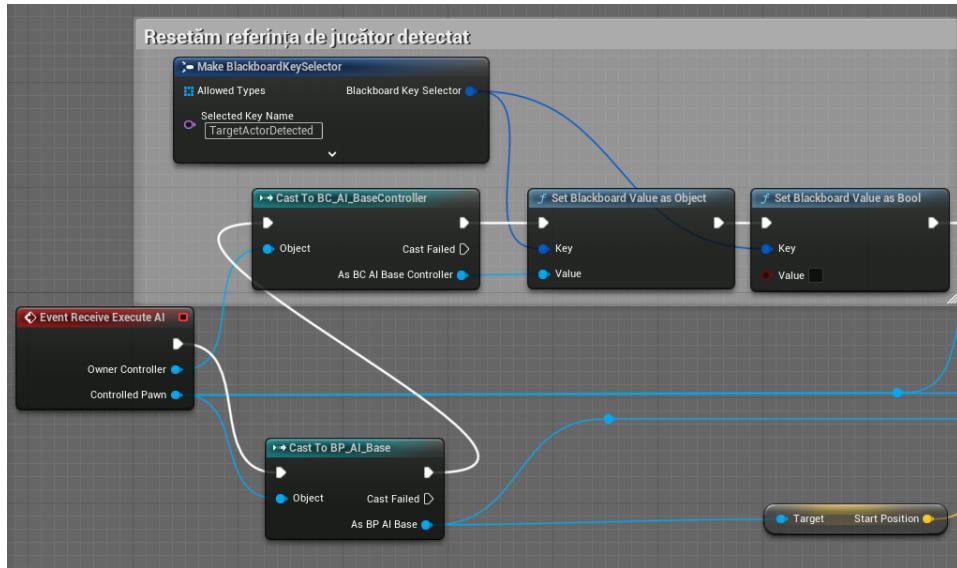


Figură 5.175 – Task-ul de dezechipare armă AI

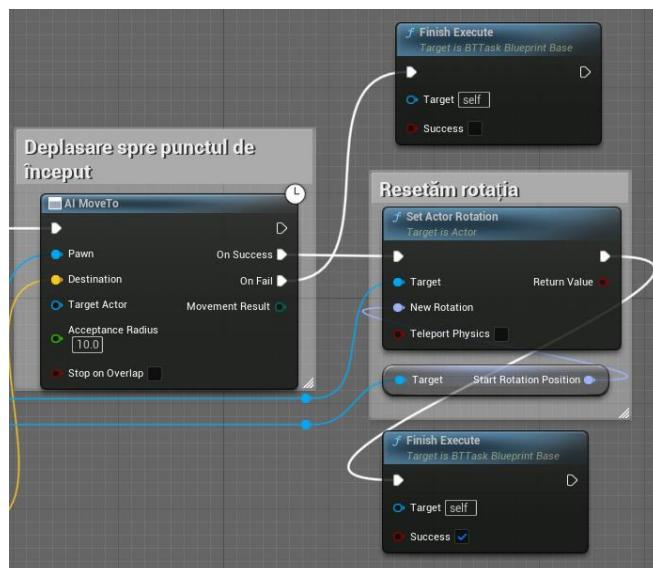


Figură 5.176 – Funcția de dezechipare armă AI

- **Task_ResetPosition** – reseteaza la poziția și rotația de start a actorului. [Figură 5.175]

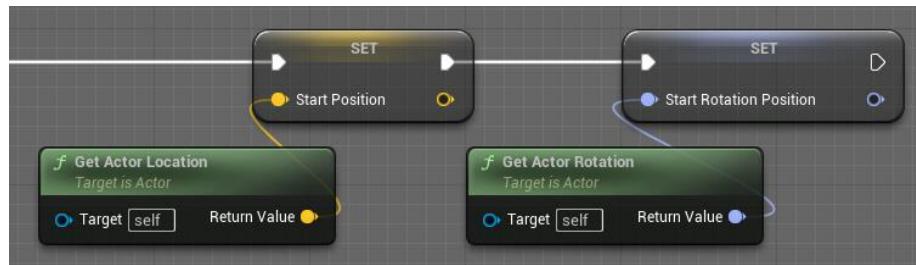


Figură 5.177 – Task-ul de resetare pozitie și rotație la start



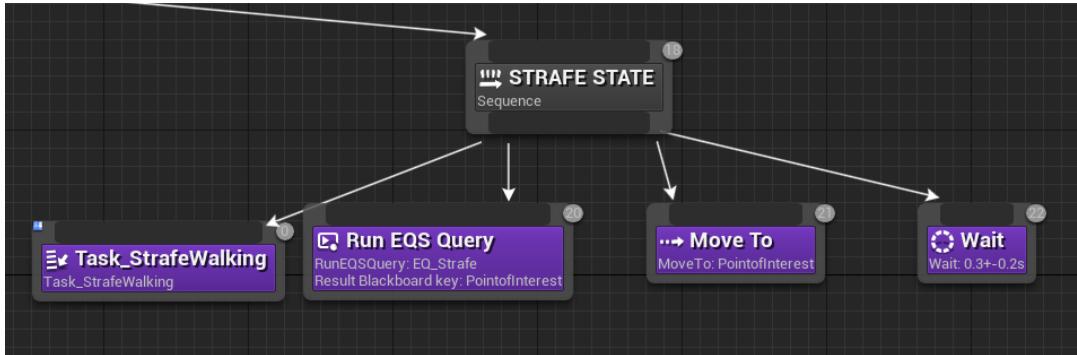
Figură 5.178 – Funcția de resetare pozitie și rotație la start

Poziția și rotația oponentului se vor salva la început de joc prin nodul de begin play astfel.



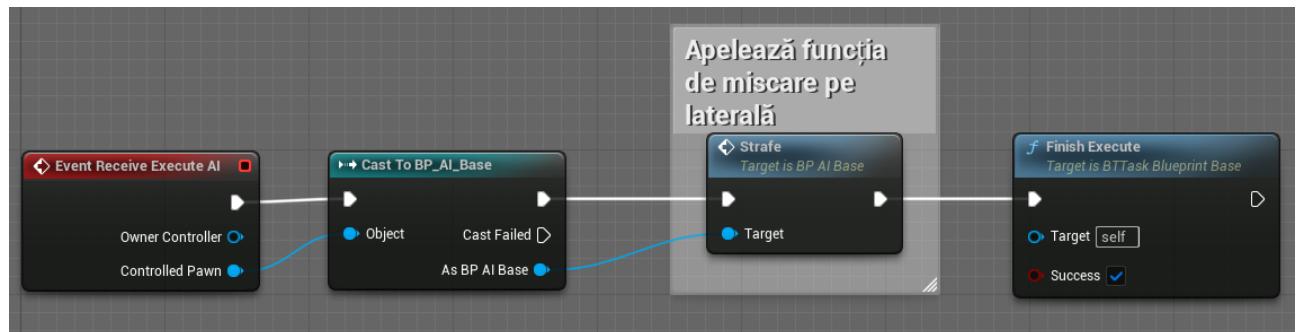
Figură 5.179 – Funcția de resetare poziție și rotație la start

Starea de strafe



Figură 5.180 – Starea de strafe

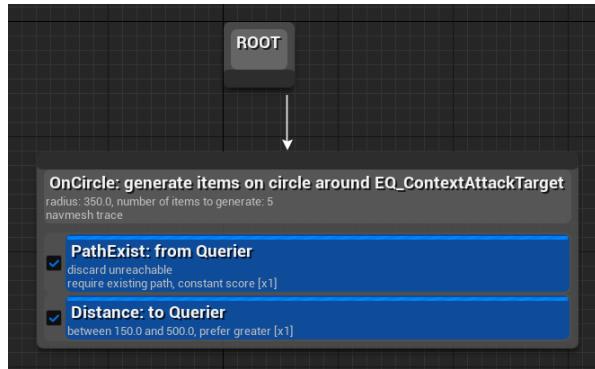
- **Task_StrafeWalking** – este un task personalizat care apelează funcția `strafe()` de mișcare pe laterală. [Figură 5.181]



Figură 5.181 – Task-ul de strafe AI

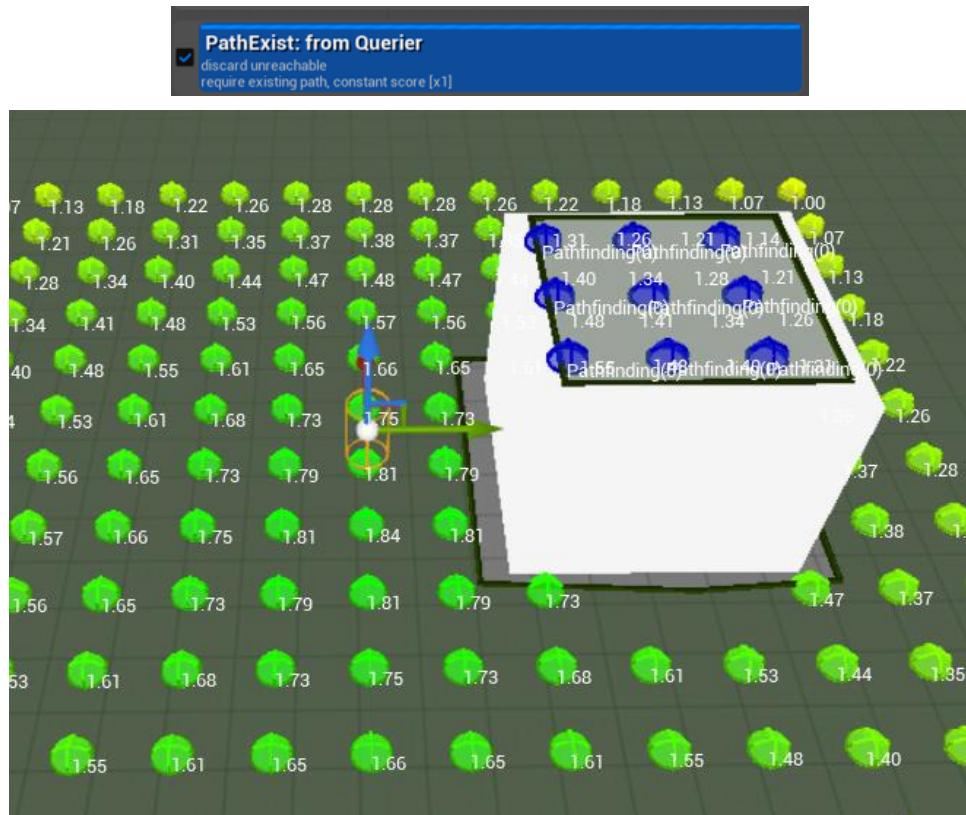
Environment Query Sistem

Environment Query System sau prescurtat, EQS este un sistem de interogare care ajută AI-ul să ia decizii în funcție de mediul înconjurător. [Figură 5.182]



Figură 5.182 – EQS

PathExist: Funcioneaza in acest mod pentru ca AI-ul să poată calcula toate posibilitățile de pathfinding din jurul jucătorului. [Figură 5.183]



Figură 5.183 – Reprezentarea vizuală de calcul al pathfinding-ului

Cele două tipuri de puncte, verzi și albastre

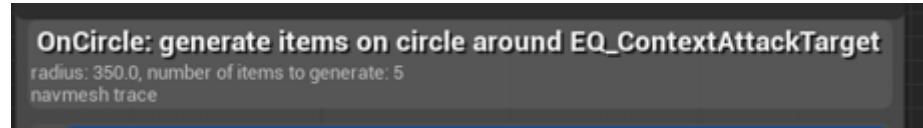
- Verde – sunt punctele pe care AI-ul **poate ajunge** și au un scor bun de EQS.
- Albastre – sunt puncte la care AI-ul **nu poate ajunge**.

Distance To Query: reprezintă test care evaluează distanța fiecărui punct generat față de Query Owner, adică jucătorul între 150 și 500 unități. [Figură 5.184]

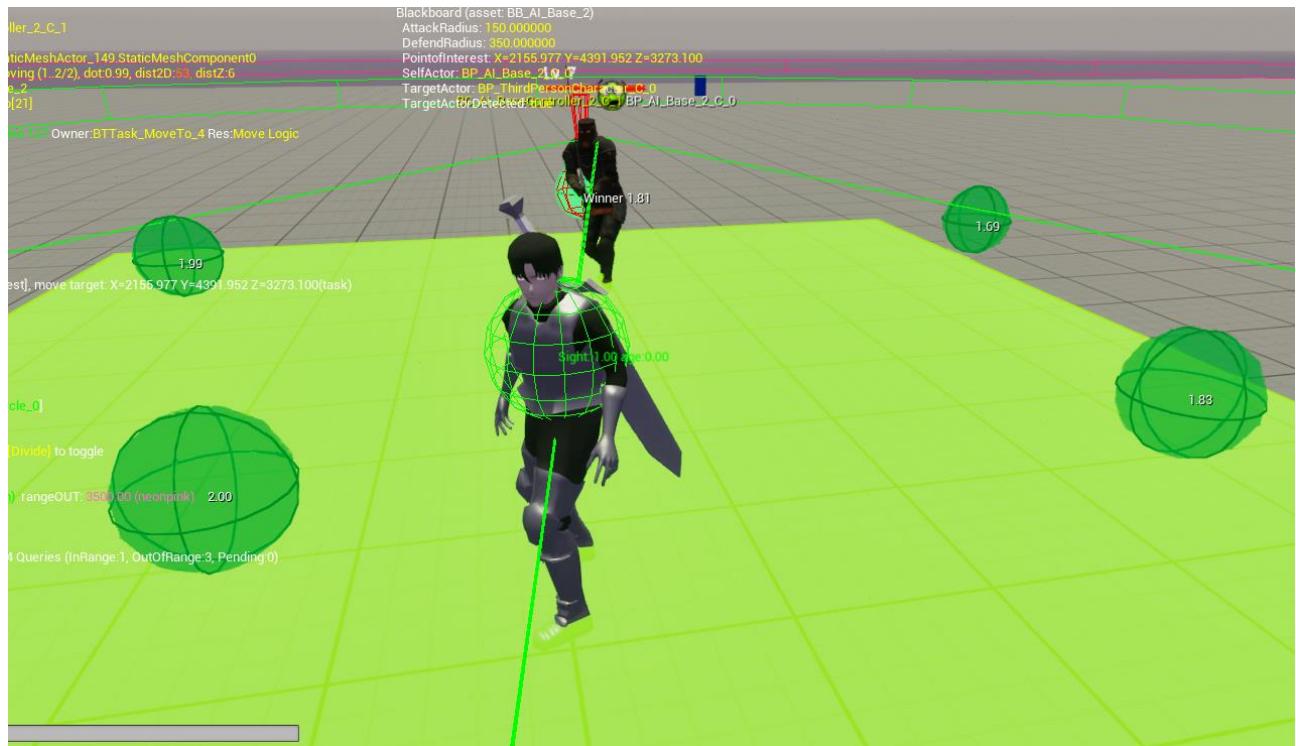


Figură 5.184 – Funcționalitatea distance to query

Funcționalitatea OnCircle: creează un cerc cu raza de 350 unități, pe care sunt plasate 5 obiecte de tip capsulă, poziționate la distanțe egale în jurul jucătorului. [Figură 5.185]



Figură 5.186 – Funcționalitatea OnCircle



Figură 5.185 – Reprezentarea vizuală a EQS de strafe

Această funcționalitate facilitează deplasarea laterală (strafe) a personajului, dintr-un punct în altul, pe traiectoria definită de cerc. Alegerea punctului de destinație se va face aleatoriu între două puncte apropiate, pentru a determina direcția de deplasare.

5.7. Sistemul de misiuni și NPC-uri pasive

Sistemul de misiuni reprezintă structura de progres al jocului prin obiective, recompense și narațiune. Acest sistem funcționează în sănsă legătură cu NPC-urile pasive, care oferă interacțiuni importante precum, atribuirea de misiuni, furnizarea de informații sau activarea anumitor evenimente din joc.

NPC-urile pasive pot fi întâlnite în diverse locații ale hărții, iar de regulă, interacțiunea jucătorului declanșează un sistem de dialoguri. Prin aceste dialoguri se pot oferi atât misiuni principale, legate de progresul general al jocului, cât și misiuni secundare. Sistemul este gestionat de un **manager de misiuni** care permite urmărirea progresului în timp real.

5.7.1. Managerul de misiuni

Managerul de misiuni este componenta centrală care gestionează întreaga logică a sistemului de misiuni. Acesta se ocupă cu **stocarea, actualizarea și monitorizarea fiecărei misiuni** pe care jucătorul o acceptă. Prin intermediul managerului UI, putem afișa misiunile active, complete sau în progres, precum și obiectivele pe care acesta trebuie să le îndeplinească.

Widgetul listei de misiuni



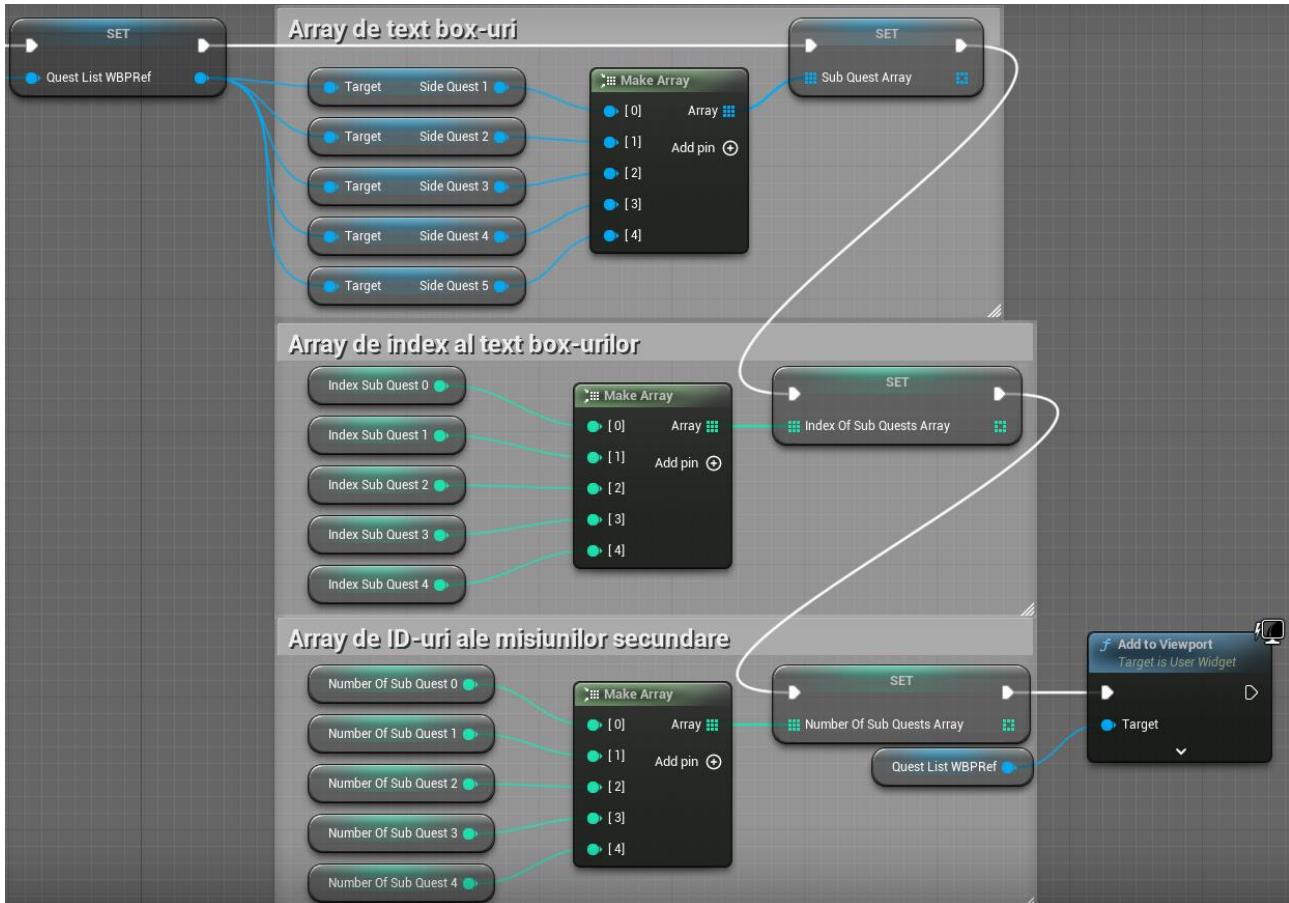
Figură 5.187 – Widgetul lista de misiuni

Managerul de misiuni este completat de o interfață vizuală care oferă jucătorului acces rapid la detaliilor de obiective curente. Această interfață este alcătuită din şase căsuțe text de tip multi-line, fiecare având rol stabilit:

- **Main Quest** – text box folosit pentru afișarea obiectivului principal de progres al jocului. Acesta reflecă misiunile importante pentru avansarea în povestea jocului.
- **Side Quest 1-5** – cinci text box-uri dedicate misiunilor secundare, care afișează detalii despre obiectivele aflate în desfășurare și condițiile necesare pentru finalizarea acestora.

Fiecare text box este actualizat automat de către managerul de misiuni la început, la fiecare progres și după finalizarea unei misiuni principale sau secundare.

Inițializarea sistemului de interfață pentru misiuni



Figură 5.188 – Inițializarea variabilelor pentru misiuni (Blueprint)

Inițializarea afișării misiunilor secundare a fost realizată prin utilizarea a trei array-uri:

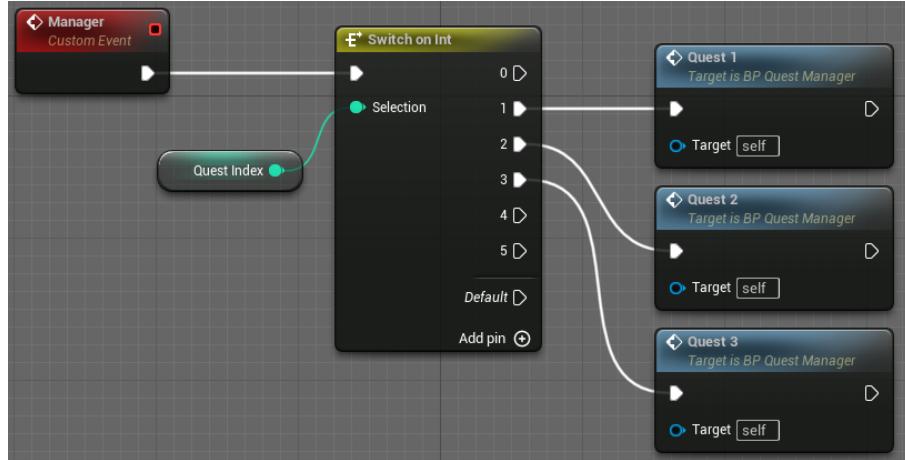
- 1. Array-ul de Text Box-uri** – conține referințele celor cinci căsuțe UI dedicate afișării misiunilor secundare.
- 2. Array-ul de Indexi** – reprezintă o listă de valori numerice care corespund pozițiilor fiecarui text box din interfață pentru a ține evidența acestora. **Toate elementele sunt inițial setate cu 0**, indicând că respectivele căsuțe **sunt libere**. Atunci când o misiune este acceptată, va fi atribuită primei căsuțe **găsită goală** iar elementul index-ului din array se va modifica în **1**, marcând astfel că **poziția este ocupată**. Această abordare facilitează **identificarea rapidă a primului slot liber pentru o nouă misiune**.
- 3. Array-ul de ID-uri pentru misiuni** – stochează identificatorii unici asociați fiecărei misiuni active. Acești identificatori sunt utilizati pentru **a accesa sau actualiza informații specifice despre o misiune**, prin intermediu managerului de gestionare al misiunilor. În momentul în care jucătorul acceptă o misiune, sistemul identifică primul index liber pe baza array-ului de indexi, iar **numărul misiunii este salvat la acel index în array-ul de ID-uri**.

5.7.2. Misiuni Principale

Adăugarea de misiuni principale

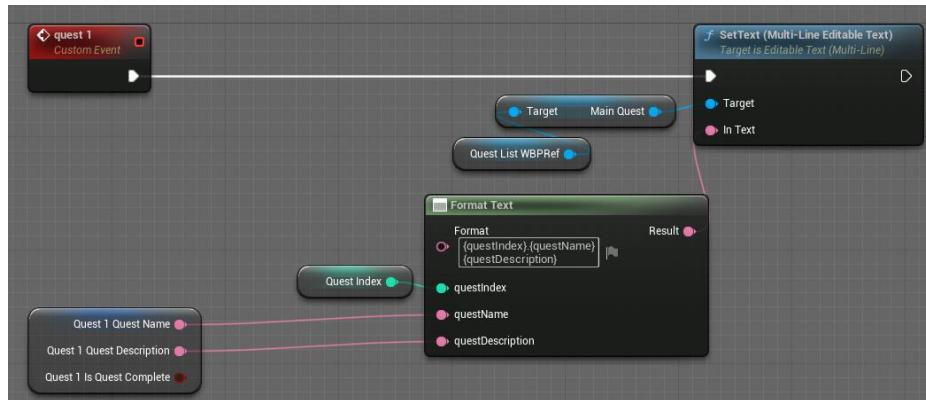
Misiunile principale (Main Quests) sunt atribuite la începutul jocului și definesc obiective esențiale pe care jucătorul trebuie să le indeplineasca pentru a progresează în poveste.

Nodul Swith on Int permite selectarea de misiuni principale pe baza unei variabile de tip integer, **questIndex**, care ține evidența misiunii curente activă. Aceasta se incrementează odată cu înăperearea misiunii curente. [Figură 5.189]



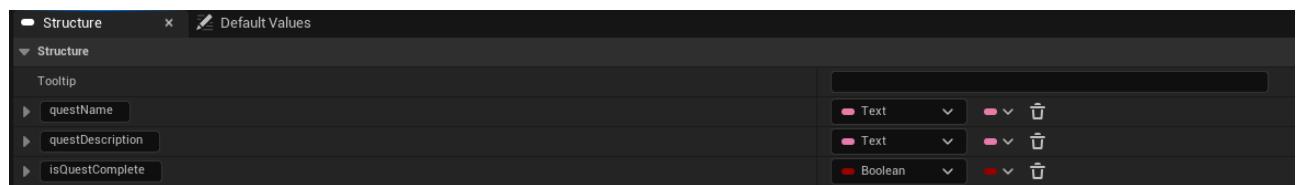
Figură 5.189 – Selectarea de misiuni principale (Blueprint)

Prin acestă logică setăm textul misiunii principale, pentru ca jucătorul să primească indicații referitoare la obiectivele pe care acesta trebuie să le completeze. [Figură 5.190]



Figură 5.190 – Adăugarea de misiuni principale (Blueprint)

Pentru afișarea textului am folosit **structuri text** care prin intermediul acestora setăm numele de misiune, descrierea, conținutul și condiții de înăpere după caz. [Figură 5.189]



Figură 5.191 – Structura de text folosită în cadrul misiunilor

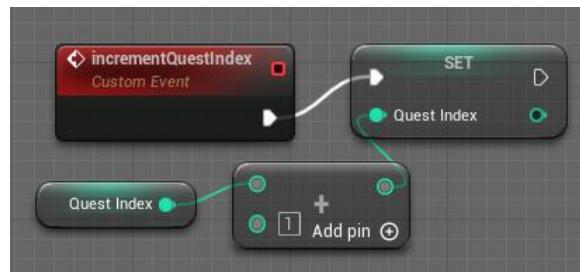
Updatarea de misiuni principale



Figură 5.192 – Updatarea de misiuni principale (Blueprint)

- Custom evenul **removeQuestList()** este folosit pentru a elimina de pe ecran.
- Custom evenul **refreshQuestList()** este folosit adăugarea pe ecran a listei de misiuni.

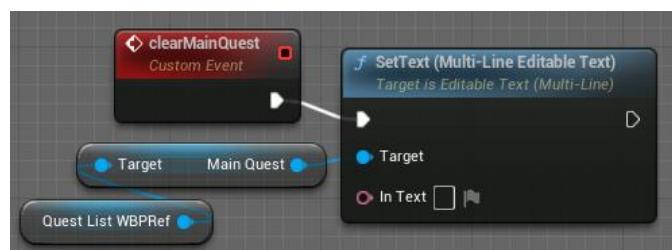
Practic cele două permit pentru o perioadă scurtă de timp nesesizabilă cu ochiul liber o tranziție prin care să actualizăm textul.



Figură 5.193 – Incrementarea numărului misiunii principale (Blueprint)

Custom evenul **incrementQuestIndex()**, este apelat în urma îndeplinirii unei misiuni pentru a începe una nouă.

Ștergerea de misiuni principale



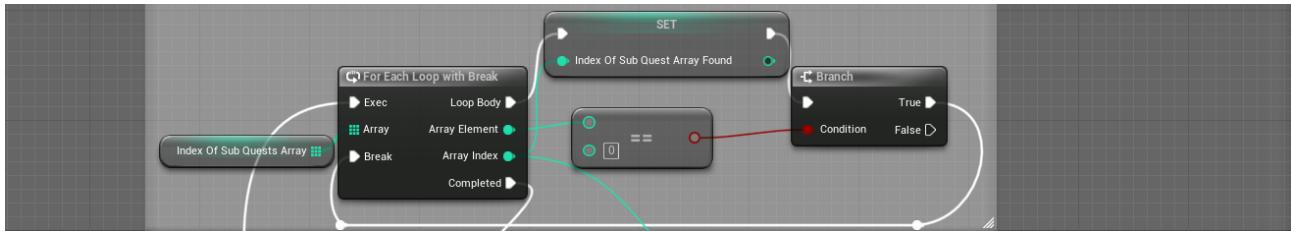
Figură 5.194 – Ștergerea de misiună principale (Blueprint)

Custom evenul **clearMainQuest()**, șterge textul misiunii principale.

5.7.3. Misiuni Secundare

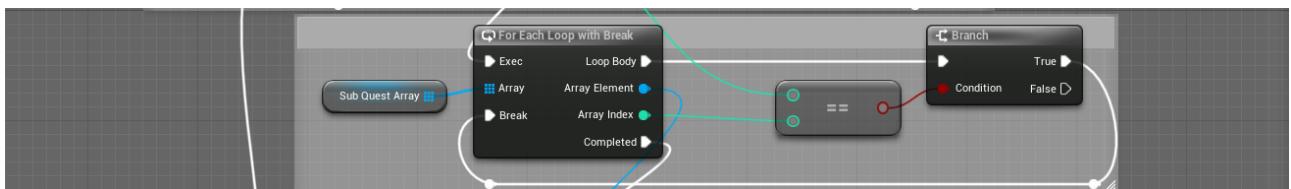
Adăugarea de misiuni secundare

- Custom evenutul **addSubquestToArray()** parcurge array-ul de indexi ai text box-urilor. Dacă am gasit cel mai apropiat index cu valoarea == 0 (liber), atunci oprim căutare și îl marcăm ca și **index găsit**. **IndexOfSubquestArrayFound** reprezintă indexul găsit și are rol de poziție comună pentru a scrie valorile în cele trei array-uri, astfel încât să rămână sincronizate pe aceeași poziții.



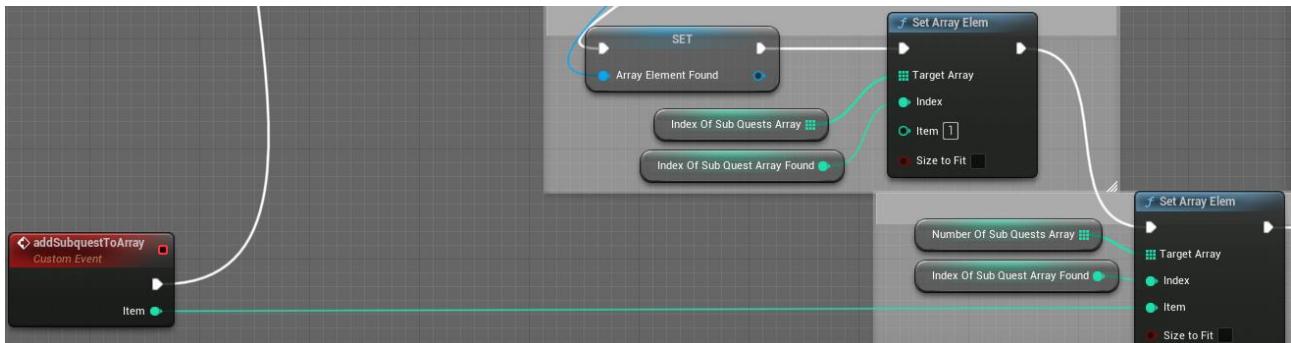
Figură 5.195 – Parcurgerea array-ului de indexi ai text box-urilor (Blueprint)

- Parcurem array-ul de referințe al text box-urilor. Dacă indexul găsit == indexul de referință al unui text box, atunci oprim căutare și setăm referința elementului găsit text box în variabila **arrayElementFound**.



Figură 5.196 – Parcurgerea array-ului de referinte ale text box-urilor (Blueprint)

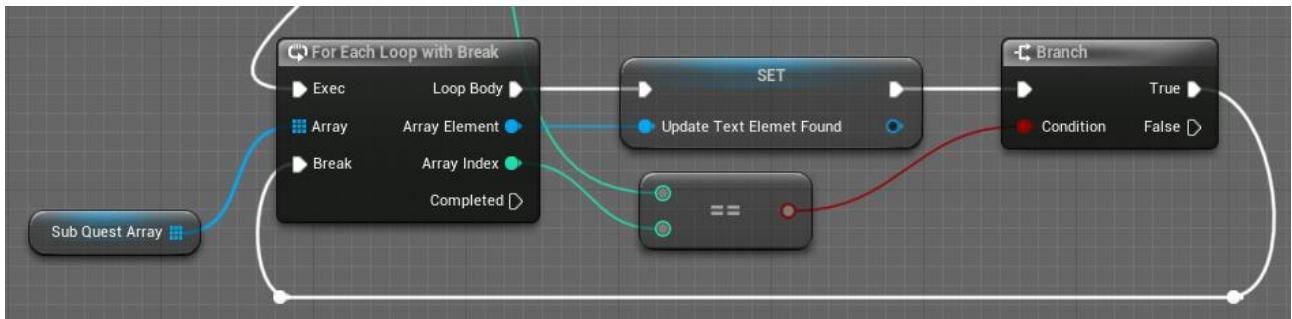
- Marcăm cu 1 (ocupat) indexul text box-ului. și cu numărul misiunii, în array-ul de ID-uri pe poziția găsită.



Figură 5.197 – Ocuparea indexelor și atribuirea de misiune (Blueprint)

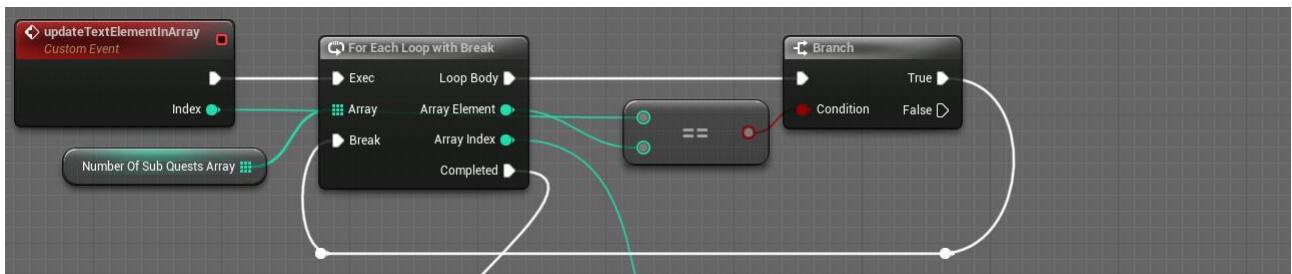
Updatarea de misiunilor secundare

- Custom evenutul **updateTextElementInArray()**, parcurge array-ul de ID-uri misiuni, dacă ID-ul corespunde cu numărul de misiune modificată, [Figură 5.198] atunci



Figură 5.198 – Parcurgem array-ul de ID-uri (Blueprint)

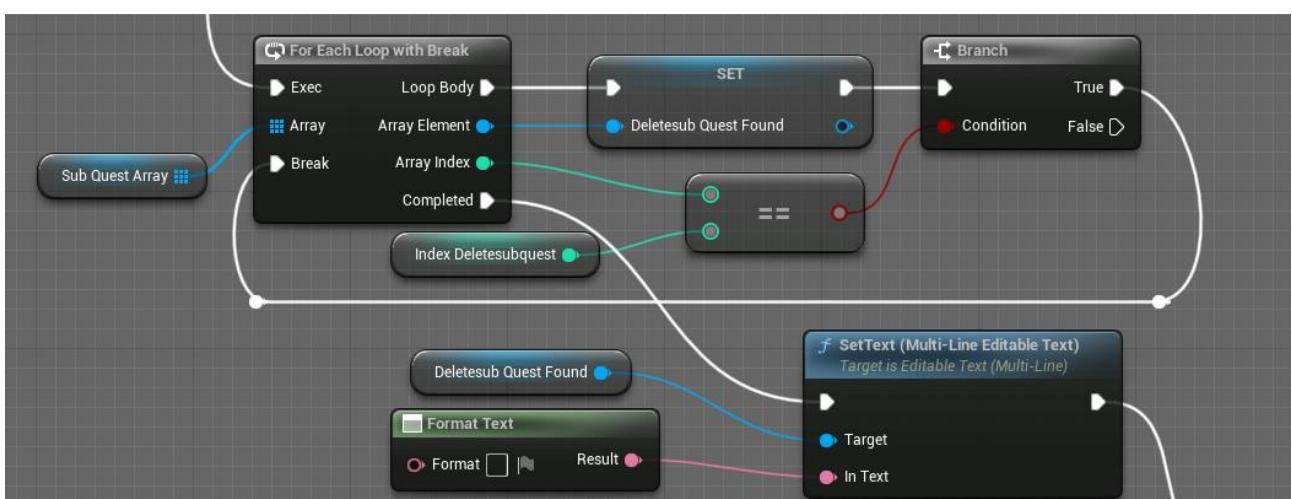
- ...Updatăm textul misiunii din text box-ul referință.



Figură 5.199 – Updatăm textul misiunii găsite(Blueprint)

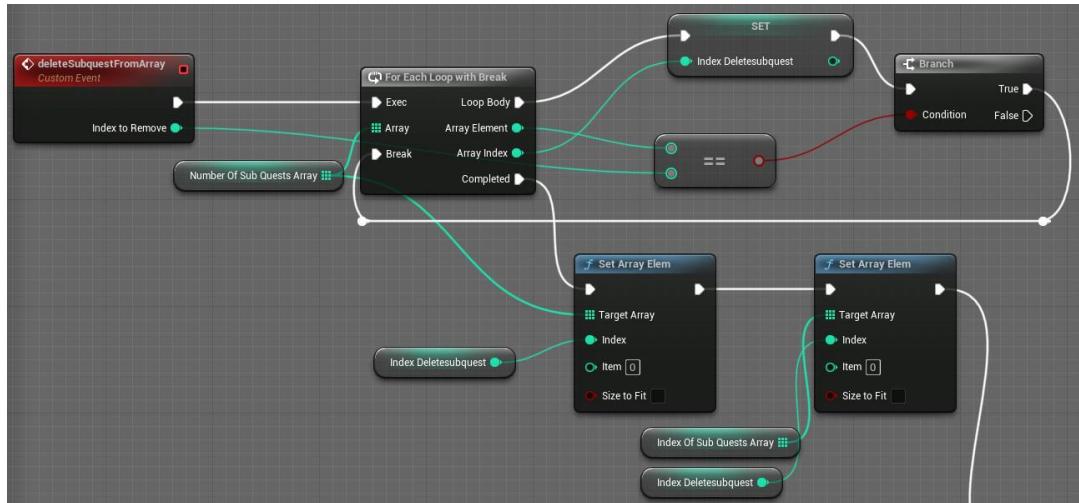
Ștergerea de misiuni secundare

- Custom event **deleteSubquestFromArray()**, parcurge tot array-ul de ID-uri. Dacă a găsit un ID cu numărul de misiune egal cu cel pe care dorim să îl ștergem, atunci va fi salvat index-ul în variabila de tip integer **indexDeleteSubquest**. Marchează cu 0 (elibereză) poziția găsită în array-ul de ID-uri misiunea. La fel și pentru array-ul de indexi ai text box-urilor.



Figură 5.200 – Parcure array ID-uri pentru misiunea completată (Blueprint)

2. Parcurge array-ul de text-boxuri, verifică dacă index-ul de ștergere este egal cu index-ul unui textbox, daca da obținem referinta text-boxului și setăm textul misiunii blank.

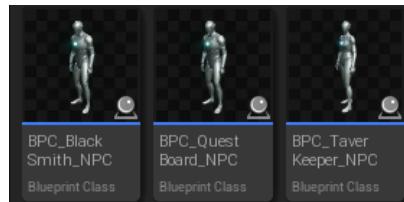


Figură 5.201 – Stergem misiunea din array-uri (Blueprint)

5.7.4 NPC-uri pasive

NPC-urile (Non-Player Character) pasive completează atmosfera generală a jocului acestea fiind de ajutor jucătorului, oferindu-i indicații sau informații utile.

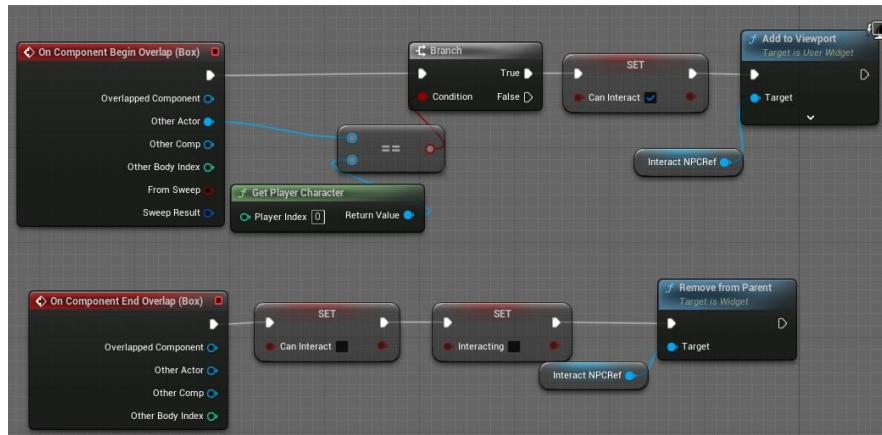
În acest proiect am realizat 3 NPC-uri de bază:



Figură 5.202 – NPC-urile utilizate (Blueprint)

- NPC Blacksmith (fierar)
- NPC Questboard (tablă de misiuni)
- NPC TavernKeeper (tavernier)

Fiecare NPC va oferi jucătorului misiuni predefinite, corespunzătoare fiecărei ramuri de progresie. În urma unei misiunilor completeate jucătorul primește recompense pe baza eforurilor depuse.



Figură 5.203 – Interacțiunea cu NPC-uri (Blueprint)

5.8. Harta de joc

Harta de joc a fost secționată în patru zone pentru a indica nivelul de complexitate, fiecare asociat cu oponenți de diferite nivele pentru a face experiența de joc una provocatoare.



Figură 5.204 – Harta de joc

6. Testarea și rezultate

6.1. Teste de performanță

Dezvoltarea, implementarea și testarea proiectului, a fost realizată pe un calculator având următoarea configurație hardware:

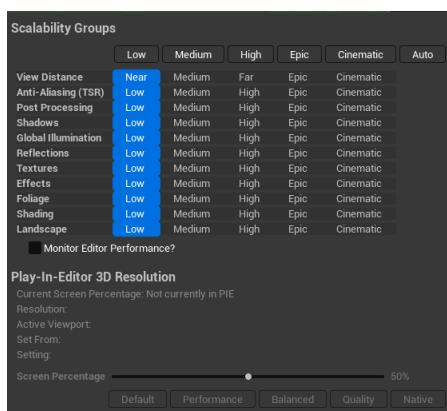
Procesor: AMD Ryzen 5 PRO

RAM: 16GB DDR4

Placă video: AMD Radeon RX 6600

SSD: 250GB

Sistem de operare: Windows 10



Figură 6.1 – Setările de scalabilitate ale calității video

Tabel de performanță

Settings/Quality	LOW	MEDIUM	HIGH	EPIC
Default	60~70 FPS	60 FPS	30 FPS	25~30 FPS
Shadow OFF	60~70 FPS	60~70 FPS	60~70 FPS	50~60 FPS
Post-Processing OFF	60~70 FPS	50~60 FPS	35 FPS	30 FPS
Shad && PP OFF	60~70 FPS	50~60 FPS	50~60 FPS	50~60 FPS
Effects	60~70 FPS	50~60 FPS	35 FPS	30 FPS
Shad && PP && Eff OFF	60~70 FPS	50~60 FPS	50~60 FPS	50~60 FPS
Textures	60~70 FPS	50~60 FPS	35 FPS	30 FPS
Shad && PP && Eff && Texture OFF	60~70 FPS	50~60 FPS	50~60 FPS	50~60 FPS
Foliage	60~70 FPS	40~50 FPS	30 FPS	27~30 FPS
Shad && PP && Eff && Texture && Foliage OFF	60~70 FPS	50~60FPS	50~60 FPS	50~60 FPS
View Distance	60~70 FPS	50~60 FPS	35 FPS	27~30 FPS
Shad && PP && Eff && Texture && Foliage && VD OFF	60~70 FPS	50~60 FPS	50~60 FPS	40~50 FPS
Anti-Aliasing (TSR)	60~70 FPS	50~60 FPS	35 FPS	27~30 FPS
Shad && PP && Eff && Texture && Foliage && VD && AA OFF ...	60~70 FPS	50~60 FPS	50~60 FPS	40~50 FPS

Tabel 6.1 – Tabelul de performanță (1)

- **Shadow (Shad)** – Controlează calitatea și detaliul umbrelor.
- **Post-Processing (PP)** – Controlează efectele: Bloom, Ambient Occlusion (umbre subtile), Motion Blur, Depth of Field, Color Grading.
- **Effect (Eff)** – Controlează calitatea efectelor de particule.
- **Texture** – Setează rezoluția texturilor folosite în scene.
- **Foliage** – Controlează câtă vegetație (ierburi, copaci, arbuști) este randată și la ce calitate.
- **View Distance (VD)** – Controlează cât de departe sunt randate obiectele din scenă (Level of Detail - LOD)
- **Anti-Aliasing (TSR - Temporal Super Resolution) (AA)** – Reduce efectul de "jagged edges" pe margini.
- **Reflections** – Controlează calitatea și tipul reflexiilor pe suprafețe lucioase (metal, sticlă, apă)
- **Landscape** – Controlează geometria terenului (dealuri, câmpii, munci).
- **Shading** – Controlează aspectul vizual al terenului

Tabelul de performanță a fost realizat în **modul Editor**. Din acest motiv, valorile de FPS sunt ușor mai scăzute față de un build standalone (unde performanța ar fi cu aproximativ +10-30% mai bună). Testele arată **impactul relativ** al fiecărei setări asupra FPS-ului.

După cum se poate observa din tabel, cele mai mari impacturi asupra performanței (FPS) sunt produse de: **Shadows, Post-Processing, Effects, Foliage și View Distance**. Pentru un minim necesar de optimizare, ar trebui dezactivate sau reduse pentru a crește performanța.

În funcție de preferințele fiecărui jucător, acesta poate alege unul dintre următoarele moduri de rulare:

- **Performance:** Priorizează rapiditatea și fluiditatea gameplay-ului, optimizând setările pentru un FPS cât mai ridicat.
- **Balanced:** Oferă un echilibru între performanță și calitate vizuală, pentru o experiență plăcută și fără compromisuri majore.
- **Quality:** Accentuează aspectul vizual și detaliile grafice, punând preț pe o imagine spectaculoasă, chiar dacă performanța poate scădea ușor.

Alte dispozitive de test ale performanței

1. **Procesor:** Intel Core i7-12700H
RAM: 16GB DDR5
Placă video: NVIDIA RTX 3070 Laptop
SSD: 512GB
Sistem de operare: Windows 10

Settings/Quality	LOW	MEDIUM	HIGH	EPIC
Default	70~80 FPS	70~75 FPS	40~50 FPS	35~40 FPS
Shadow OFF	70~80 FPS	70~80 FPS	60~70 FPS	50~60 FPS
Post-Processing OFF	70~80 FPS	65~75 FPS	35 FPS	30 FPS
Shad & PP OFF	70~80 FPS	65~75 FPS	50~60 FPS	50~60 FPS
Effects	70~80 FPS	65~75 FPS	35 FPS	30 FPS
S & PP & Eff OFF	70~80 FPS	65~75 FPS	50~60 FPS	50~60 FPS
Textures	70~80 FPS	65~75 FPS	35 FPS	30 FPS

S & PP && Eff && Texture OFF	70~80 FPS	65~75 FPS	50~60 FPS	50~60 FPS
Foliage	70~80 FPS	55~65 FPS	30 FPS	27~30 FPS
S & PP & Eff & Texture & Foliage OFF	70~80 FPS	65~75 FPS	50~60 FPS	50~60 FPS
View Distance	70~80 FPS	65~75 FPS	35 FPS	27~30 FPS
S & PP & Eff & Texture & Foliage & VD OFF	70~80 FPS	65~75 FPS	50~60 FPS	40~50 FPS
Anti-Aliasing (TSR)	70~80 FPS	65~75 FPS	35 FPS	27~30 FPS
S & PP & Eff & Texture & Foliage && VD && AA OFF	70~80 FPS	65~75 FPS	50~60 FPS	40~50 FPS
...				

Tabel 6.2 – Tabelul de performanță (2)

2. Procesor: Ryzen 5 1600x

RAM: 16GB DDR4

Placă video: AMD RX 6650XT

SSD: 1TB

Sistem de operare: Windows 11

Settings/Quality	LOW	MEDIUM	HIGH	EPIC
Default	70~80 FPS	70~75 FPS	40~50 FPS	35~40 FPS
Shadow OFF	70~80 FPS	70~80 FPS	60~70 FPS	50~60 FPS
Post-Processing OFF	70~80 FPS	65~75 FPS	35 FPS	30 FPS
Shad && PP OFF	70~80 FPS	65~75 FPS	50~60 FPS	50~60 FPS
Effects	70~80 FPS	65~75 FPS	35 FPS	30 FPS
Shad && PP && Eff OFF	70~80 FPS	65~75 FPS	50~60 FPS	50~60 FPS
Textures	70~80 FPS	65~75 FPS	35 FPS	30 FPS
Shad && PP && Eff && Texture OFF	70~80 FPS	65~75 FPS	50~60 FPS	50~60 FPS
Foliage	70~80 FPS	55~65 FPS	30 FPS	27~30 FPS
S && PP && Eff && Texture && Foliage OFF	70~80 FPS	65~75 FPS	50~60 FPS	50~60 FPS
View Distance	70~80 FPS	65~75 FPS	35 FPS	27~30 FPS
Shad & PP & Eff & Texture & Foliage & VD OFF	70~80 FPS	65~75 FPS	50~60 FPS	40~50 FPS
Anti-Aliasing (TSR)	70~80 FPS	65~75 FPS	35 FPS	27~30 FPS
Shad & PP & Eff & Texture & Foliage & VD & AA OFF ...	70~80 FPS	65~75 FPS	50~60 FPS	40~50 FPS

Tabel 6.3 – Tabelul de performanță (3)

6.2. Probleme întâmpinate și soluții

Implementarea blending-ului pentru textura Landscape-ului

Problema: Nu am găsit o soluție clară pentru a realiza un material dinamic de tip seasonal landscape, care să permită schimbarea anotimpurilor prin combinarea texturilor.

Soluție: Am creat un sistem personalizat, în care am separat funcțiile de texturare pe anotimpuri (primăvară, vară, toamnă, iarnă), iar cu ajutorul nodului Lerp și a unui Material Parameter Collection, am combinat aceste texturi condiționat, pe baza unor parametri binari de control, conformat [Tabel 5.6]

- 0 0 0 – anotimpul de vară
- 0 0 1 – anotimpul de primăvară
- 0 1 0 – anotimpul de toamnă
- 1 0 0 – anotimpul de iarnă

Probleme de detectie multiplă a coliziunii pentru atacurile inamicilor

Problema: În situația în care mai mulți inamici erau poziționați foarte aproape unul de celălalt, sistemul de detectie a coliziunii înregistra un număr incorrect de lovitură — primul inamic primea un singur hit, al doilea două lovitură, iar al treilea chiar trei hituri pentru un singur atac.

Soluție: Am implementat o condiție suplimentară care validează coliziunea doar la primul contact și blochează detectarea ulterioară de lovitură până la finalizarea animației de atac. La finalul acesteia, sistemul resetează validarea coliziunilor, permitând reluarea normală a atacurilor conform [Figură 5.146]

Calculul greșit al experienței (XP)

Problema: La obținerea unei cantități foarte mari de XP (ex: 3000), algoritmul de creștere în nivel (level-up) nu păstra experiența în exces după primul nivel. Astfel, după un level-up, experiența era resetată iar restul era pierdut.

Soluție: Am rezolvat problema folosind un algoritm recursiv, care continuă să aloce XP-ul rămas și să crească nivelul până când întreaga cantitate de XP a fost procesată corect conform [Figură 5.64], [Figură 5.65], [Figură 5.66]

7. Contribuții proprii

- **Sistemul de timp**

Am implementat un sistem de timp care contorizează orele, minutele și secundele în timp real în cadrul jocului. Acesta controlează tranziția vizuală între momentele zilei (zi, apus, noapte, răsărit) prin animarea poziției soarelui și a lunii și modificarea parametrilor vizuali.

- **Sistemul de anotimpuri**

Am implementat patru anotimpuri (primăvară, vară, toamnă, iarnă), prin posibilitatea de schimbarea a texturilor și culorilor mediului în funcție de anotimpul activ. Schimbarea se face gradual, prin blend de texturi și materiale parametrizate.

- **Sistemul de fenomene meteorologice**

Am dezvoltat un sistem de fenomene meteorologice dinamice care utilizează efecte vizuale realizate cu Niagara Effects (sistemul de particule din Unreal Engine).

Sistemul permite apariția și combinarea mai multor tipuri de condiții atmosferice, precum:

- ploaie;
- ninsoare;
- ceată.

Aceste efecte pot fi activate manual, pentru a crea o atmosferă specifică în funcție de preferințele de joc ale utilizatorului. De asemenea, este posibilă suprapunerea efectelor rezultând un mediu de joc mai variat și mai imersiv.

- **Sistemul de îmbunătățire atribute**

Am implementat un sistem de atribute care permite îmbunătățirea acestora, precum și funcționalitatea de resetare (respec), oferind astfel jucătorului flexibilitate în personalizarea personajului.

- **Sistemul de schimb al înfățișării**

Am implementat un sistem care permite modificarea componentelor RGB ale armurii personajului principal, oferind astfel posibilitatea de personalizare vizuală în funcție de preferințele jucătorului.

- **Sistem de gestionare al misiunilor**

Am implementat un sistem de misiuni bazat pe utilizarea array-urilor, pentru afișarea și urmărirea obiectivelor pe care jucătorul trebuie să le îndeplinească.

Structura sistemului:

- Un array de căsuțe (text box-uri), în care se afișează informațiile legate de obiectivele active;
- Un array de indexuri pentru a urmări starea fiecărei căsuțe (0 — căsuță liberă, 1 — căsuță ocupată cu o misiune activă);
- Un array de ID-uri de misiuni, care stochează identificatorul unic al fiecărei misiuni pe indexul corespunzător.
- Actualizarea și ștergerea misiunilor se face prin resetarea valorilor corespunzătoare din array-uri (index și ID), astfel încât să poată fi reutilizate pentru noi misiuni.

- **Implementarea de oponenți variați în funcție de nivel**

8. Concluzii

În concluzie, am dezvoltat un joc de acțiune 3D de tip Fantasy RPG utilizând Unreal Engine, concentrându-ne pe integrarea elementelor definitorii ale acestui gen. Pe parcursul elaborării lucrării, am reușit să construiesc o bază solidă și funcțională, care poate servi drept punct de plecare pentru dezvoltări viitoare.

Lucrul cu sistemul Blueprint a reprezentat pentru mine o experiență nouă și provocatoare. Deși la început părea mai puțin intuitiv, pe măsură ce am aprofundat cunoștințele și am lucrat la diverse mecanici, am reușit să înțeleg logica detaliată și să dezvolt funcționalități variate, care contribuie la o experiență de joc coerentă și captivantă.

Acest proiect m-a ajutat să înțeleg mai bine procesul complex de dezvoltare a unui joc, în special modul în care sunt structurate clasele, moștenirile și obiectele în cadrul unui motor de joc precum Unreal Engine. Utilizarea sistemului vizual Blueprint mi-a oferit o perspectivă clară asupra modului în care componentele interacționează și asupra fluxului de execuție într-un proiect cu mai multe elemente interdependente.

De asemenea, am dobândit o mai bună înțelegere a importanței optimizării și a modularității codului în dezvoltarea de jocuri, precum și a problemelor tehnice care pot apărea pe parcurs. Consider că această experiență a contribuit semnificativ la dezvoltarea mea profesională și la îmbunătățirea abilităților tehnice, oferindu-mi o bază solidă pentru proiectele viitoare.

Pe viitor, aş dori să extind proiectul prin integrarea unor funcționalități suplimentare, cum ar fi:

- un sistem mai avansat de inteligență artificială pentru inamici
- un sistem de inventar și echipamente
- poveste interactivă ramificată
- mecanici de progresie complexă a personajului (leveling)
- efecte de sunete
- optimizare generală a performanței.

Bibliografie

- [1] https://en.wikipedia.org/wiki/Role-playing_game – accesat în 10 iunie 2025
- [2] <https://en.wikipedia.org/wiki/Fantasy> – accesat în 10 iunie 2025
- [3] <https://en.bandainamcoent.eu/dark-souls/dark-souls> – accesat în 10 iunie 2025
- [4] <https://baldursgate3.game/> – accesat în 10 iunie 2025
- [5] <https://www.thewitcher.com/ro/en/witcher3> – accesat în 10 iunie 2025
- [6] https://en.wikipedia.org/wiki/Game_engine – accesat în 10 iunie 2025
- [7] <https://www.unrealengine.com/en-US> – accesat în 10 iunie 2025
- [8] <https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine> – accesat în 10 iunie 2025
- [9] <https://krita.org/en/> – accesat în 10 iunie 2025
- [10] <https://www.blender.org/> – accesat în 10 iunie 2025
- [11] <https://www.mixamo.com/#/> – accesat în 10 iunie 2025
- [12] <https://www.fab.com/> – accesat în 10 iunie 2025
- [13] <https://www.fab.com/listings/b066de06-73b8-4fbe-b30c-468f5bcf7575> – accesat în 10 iunie 2025
- [14] <https://www.fab.com/listings/e1486393-20eb-4b8a-8bb1-e87ed04f36f3> – accesat în 10 iunie 2025
- [15] <https://dev.epicgames.com/documentation/en-us/unreal-engine/overview-of-blueprints-visual-scripting-in-unreal-engine> – accesat în 10 iunie 2025
- [16] <https://dev.epicgames.com/documentation/en-us/unreal-engine/custom-events-in-unreal-engine> – accesat în 10 iunie 2025
- [17] <https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprint-interface-in-unreal-engine> – accesat în 10 iunie 2025
- [18] <https://dev.epicgames.com/documentation/en-us/unreal-engine/widget-blueprints-in-umg-for-unreal-engine> – accesat în 10 iunie 2025
- [19] https://en.wikipedia.org/wiki/Concept_art – accesat în 10 iunie 2025
- [20] <https://dev.epicgames.com/documentation/en-us/unreal-engine/rigging-with-control-rig-in-unreal-engine> – accesat în 10 iunie 2025
- [21] <https://dev.epicgames.com/documentation/en-us/unreal-engine/animation-blueprints-in-unreal-engine> – accesat în 10 iunie 2025
- [22] <https://dev.epicgames.com/documentation/en-us/unreal-engine/state-machines-in-unreal-engine> – accesat în 10 iunie 2025
- [23] https://dev.epicgames.com/documentation/en-us/unreal-engine/blend-spaces?application_version=4.27 – accesat în 10 iunie 2025
- [24] <https://dev.epicgames.com/documentation/en-us/unreal-engine/animation-slots-in-unreal-engine> – accesat în 10 iunie 2025
- [25] <https://dev.epicgames.com/documentation/en-us/unreal-engine/animation-montage-editor-in-unreal-engine> – accesat în 10 iunie 2025
- [26] <https://dev.epicgames.com/documentation/en-us/unreal-engine/artificial-intelligence-in-unreal> – accesat în 10 iunie 2025
- [27] <https://dev.epicgames.com/documentation/en-us/unreal-engine/landscape-quick-start-guide-in-unreal-engine> – accesat în 10 iunie 2025
- [28] <https://dev.epicgames.com/documentation/en-us/unreal-engine/using-sockets-with-static-meshes-in-unreal-engine> – accesat în 10 iunie 2025
- [29] <https://dev.epicgames.com/documentation/en-us/unreal-engine/BlueprintAPI/Game/Damage/ApplyDamage> – accesat în 10 iunie 2025
- [30] <https://dev.epicgames.com/documentation/en-us/unreal-engine/BlueprintAPI/AddEvent/Game/Damage/EventAnyDamage> – accesat în 10 iunie 2025

- [31] https://dev.epicgames.com/documentation/en-us/unreal-engine/niagara-overview?application_version=4.27 – accesat în 10 iunie 2025
- [32] <https://dev.epicgames.com/documentation/en-us/unreal-engine/ai-controllers-in-unreal-engine> – accesat în 10 iunie 2025
- [33] <https://dev.epicgames.com/community/learning/tutorials/Mldp/introduction-to-blackboard-and-behavior-trees> – accesat în 10 iunie 2025