# Raport de Testare Manuala - B3 Catan Testing

## Mediu de testare

-OS: Linux Mint 21.3 Virginia (Kernel 5.15.0-91-generic, x86\_64)

- Unity Editor: 6000.0.43f1 LTS

- Scenă testată: Game.unity

- IDE: Rider (Attach to Unity Editor)

## 1. Initial Placement

La pornirea jocului, fiecare jucător trebuie să plaseze câte o așezare și un drum în ordinea crescătoare a indexului (0→1→2→3), apoi în ordinea descrescătoare (3→2→1→0).

În metoda `StartMachineWithOtherPlayer()`, apelul `StartMachine()` repornea `InitialiseState`, regenerând tabla si reinițializând jucătorii. Acest comportament crea un blocaj și crash după primul EndTurn.

Rezolvare:

În `StartMachineWithOtherPlayer()

am înlocuit: StartMachine();

cu stateMachine.ChangeState<AddHouseState>();

Astfel, nu se mai repornește fluxul de inițializare, iar plasarea inițială se efectueaza corect.

**2. Ordinea Jucatorilor dupa PlayerEndState**

După faza de plasare inițială (0→1→2→3→3→2→1→0), jocul trebuie sa reia ciclul normal (0→1→2→3). Din cauza omiterii reinițializării variabilei \_isInverseOrder, din methoda **AddRoadState.OnEnter** în blocul if:

if (stateMachine.IsFirstPlayer && \_isInverseOrder)

{

stateMachine.ChangeState<PlayerDiceGameState>();

yield break;

}

ordinea jucătorilor nu era eronată. Ca rezolvare, am completat condiția pentru a reseta flag-ul `**\_isInverseOrder**` și a reveni la ordinea normală

## 3. Distributie Resurse (roll ≠ 7)

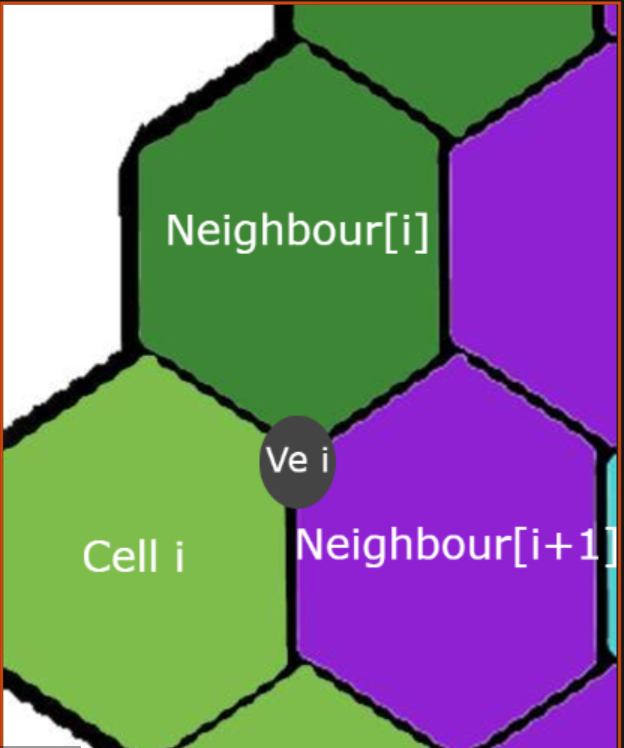
După aruncarea zarurilor a căror sumă este o valoare diferită de 7, jocul intră în **ResourceDistributionState** și distribuie resurse jucatorilor cu așezări lângă hex-urile corespunzătoare. Fluxul continuă apoi natural spre `**PlayerFreeState`.**

## 4. Construirea asezarilor si drumurilor (Building Logic)

Metodele `**BuildHouseCoroutine**` cât si **BuildRoadCoroutine** din clasa **HumanPlayer** folosesc metodele din clasa `**FullHexGrid**` pentru a obtine vertex-uri ,muchii și vecini.   
 În încercarea de a construi o casă prin metoda BuildHouseCoroutine se observă că settlement-urile sunt plasate eronat cu o probabilitate de ⅙. În **BuildHouseCoroutine** este apelată metoda RayCastCoroutine care are rolul de a trage o rază imaginarp de la cameră la punctul în care a fost apăsat și a găsi cel mai apropiat obiect de acea rază. Având obiectul găsit(va fi un hex) se caută prin metoda **getClosestCorner** cel mai apropiat colț al celulei lovite de clickul efectiv(**\_hitpoint**). Prin Loguri de debug se observă că var corners = boardGrid.GetHexVertices(hexPosition) oferă un tuplu de settlement și direcția propriu-zisă a colțului hexagonului(dreapta sus ,dreapta,dreapta jos,...) în care poziția settlementului e mereu duplicată si nu coinicde întotdeauna cu  
var cornerPosition = boardGrid.GetHexCorner(corner.Item2, hexPosition);(corner.item2 e direcția colțului hexagonului)  
 Mergând în **FullHexGrid**, în metoda public Cell this[HexPosition position] , unde sunt initialization celulele și proprietățile lor găsim for-ul acesta care caută să folosească vertexuri și edgerui ale unor vecini deja existenți   
for (int i = 0; i < 6; i++) {

edges[i] = neighbours[i]?.Edges[(i + 3) % 6];

vertices[i] = (neighbours[i]?.Vertices[(i + 2) % 6]) ?? (neighbours[(i+1) % 6]?.Vertices[(i + 3) % 6]);

}  


Logica de reutilizarea a vertex-lui vecinilui de la poziti i+1 nu era corectă `(i+3)%6`, ducând la duplicare și/sau omiterea unor colturi valide. În celula vecina i+1 pozitia nodului corespunzător de intersecție nu este i+3, ci i+4 pentru a ajunge la corespondentul corect în plan.

Rezolvare:

În **`FullHexGrid`**:

vertices[i] = neighbours[i]?.Vertices[(i + 2) % 6]

?? neighbours[(i + 1) % 6]?.Vertices[(i + 4) % 6];

## 6. Buff-uri Porturi

Există 2 tipuri de porturi

- Port universal → buff 3:1

- Port specific → buff 2:1 pentru o resursă specifică

Folosind un log de debuger am verificat adăugarea corectă a efectelor de player buff pentru comercializarea cu banca în cazul adăugărei unui settlement într-un hexagon cell care este de tipul port și omiterea oricărui efect în caz contrar

In `AddPortBuffForSettlement`:

string buffText = port.ResourceType.HasValue

? $"2:1 {port.ResourceType.Value}"

: "3:1";

Debug.Log($"[PORT DEBUG] Applying port buff {buffText} to player {player.name}");

## 7. Validare Distribuire Resurse

După adăugarea building-urilor și road-urilor, jocul face tranziția către PlayerDiceState care simulează aruncarea zarurilor de jucător cu un generator random. Bazat pe suma punctelor celor 2 zaruri, dacă este diferită de 7, jocul distribuie corect resurse jucătorilor care au construit settlement uri pe un hexagon/celulă cu numărul = cu suma respectivă.

Pe lângă debugging , logurile de verificare din `**ResourceDistributionState**` pe care le-am adăugat confirmă corectitudinea funcționalității

Debug.Log($"Resource Given to: {[owner.name](http://owner.name)}");

## 8. Pași următori

## -testarea ThiefController și ThiefState

## -testarea butoanelor front-endului pentru trading with players + with bank , buying development cards

## -testarea logicii AI-ului de schimbare de rundă trading plasamente obiecte