

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR,
INFORMATIKA SZAK**



SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM

A logika varázsa: a puzzle típusú játékok készítésének lépései

DIPLOMADOLGOZAT

Témavezető:
Dr. Osztián Erika,
Egyetemi adjunktus
Osztián Pálma-Rozália,
Egyetemi tanársegéd

Végzős hallgató:
Bodor Benjamin

2023

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
SPECIALIZAREA INFORMATICĂ



UNIVERSITATEA
SAPIENTIA

Magia logicii: pași pentru a crea jocuri puzzle

LUCRARE DE DIPLOMA

Coordonator științific:
Dr. Osztián Erika,
Lector universitar
Osztián Pálma-Rozália,
Asistent universitar

Absolvent:
Bodor Benjamin

2023

**SAPIENTIA HUNGARIAN UNIVERSITY OF
TRANSYLVANIA
FACULTY OF TECHNICAL AND HUMAN SCIENCES
COMPUTER SCIENCE SPECIALIZATION**



SAPIENTIA
HUNGARIAN UNIVERSITY
OF TRANSYLVANIA

The magic of logic: steps to make puzzle games

BACHELOR THESIS

Scientific advisor:
Dr. Osztián Erika,
Lecturer
Osztián Pálma-Rozália,
Assistant professor

Student:
Bodor Benjamin

2023

LUCRARE DE DIPLOMĂ

Coordonator științific:
Lect. univ. dr. Osztian Erika

Candidat: **Bodor Benjamin**
Anul absolvirii: **2023**

a) Tema lucrării de licență:

Proiectarea și dezvoltarea elementelor de puzzle și animații pentru un joc educativ 3D care oferă o experiență plăcută utilizatorului

b) Problemele principale tratate:

Realizarea unui studiu bibliografic privind animațiile caracterilor și elementelor puzzle din jocurile 2D.

Studiul motorului de joc Unity și a tehnologiilor oferite de acesta.

Proiectarea și realizarea unor animații și elemente puzzle bazate pe aceste informații.

c) Desene obligatorii:

Diagrame de proiectare pentru aplicația software realizată

d) Softuri obligatorii:

În timpul dezvoltării jocului în Unity, au fost create animațiile necesare pentru mișcarea personajului principal, precum și acțiuni specifice. De asemenea, a fost elaborat un sistem de indicii și diverse elemente de puzzle, care contribuie la o experiență plăcută a utilizatorului.

e) Bibliografia recomandată:

- Shokeen E, PELLICONE A, WEINTROP D, KETELHUT D, CUKIER M, PLANE J and WILLIAMS-PIERCE C. Children's Approaches to Solving Puzzles in Videogames. SSRN Electronic Journal. 10.2139/ssrn.4130855.
- Dong, T., Dontcheva, M., Joseph, D., Karahalios, K., Newman, M., & Ackerman, M. (2012, May). Discovery-based games for learning software. In Proceedings of the SIGCHI conference on human factors in computing systems (pp. 2083-2086).
- Joshi A, Mousas C, Harrell D and Kao D. Exploring the Influence of Demographic Factors on Progression and Playtime in Educational Games. Proceedings of the 17th International Conference on the Foundations of Digital Games. (1-15).

f) Termene obligatorii de consultații: lunar, preponderent online

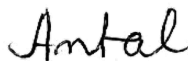
g) Locul și durata practicii: Universitatea „Sapientia” din Cluj-Napoca,
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, laboratorul de informatică (sala 415)
Primit tema la data de: 20.05.2022.

Termen de predare: 02.07.2023.

Semnătura Director Departament



Semnătura responsabilului
programului de studiu



Semnătura coordonatorului



Semnătura candidatului



Declarație

Subsemnatul/a Bodor Benjamin, absolvent(ă) al/a specializării
INFORMATICA, promoția 2023, cunoscând
prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a
Universității Sapienția cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare
de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea
este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în
mod corespunzător.

Localitatea,

Data:

TÂRGU MUREȘ

14.06.2023.

Absolvent

Semnătura Bodor

Kivonat

A játékok felhasználói élményét és motivációját meghatározó elemek közé tartoznak a puzzle játékok, animációk, vizualizációk, interakciók és maga a karakter tervezés. Ezek a fontos komponensek nemcsak esztétikailag vonzóvá teszik a játékot, hanem aktív részvételt és élvezetet biztosítanak a játékosnak.

A puzzle játékok izgalmas kihívásokat jelentenek, melyek megoldásához kreatív gondolkodásra és problémamegoldó képességekre van szükség. A változatos feladatok logikai lépésekre ösztönzik a játékosot, melyek során össze kell kapcsolnia és összehangolnia a különböző elemeket vagy tárgyakat a megoldás eléréséhez.

Az animációk és vizualizációk dinamikusságot és élettelséget kölcsönöznek a játéknak. Ezek a mozdulatok, átmenetek és látványos effektek valóságosabbá és lenyűgözőbbé teszik a játék világát, így lekötik a játékos figyelmét. Az animációk segítségével a karakterek és környezetük mozgásban vannak, ami fokozza a játék interaktivitását és élvezetességét.

Az interakció az egyik legfontosabb elem a játékelményben, hiszen lehetővé teszi a játékos aktív részvételét a játék világában. A játékos cselekedetei és választásai hatással vannak a játékmenetre és a történetre. Az interakció lehetőséget ad arra, hogy a játékos irányítsa a karaktert, manipulálja a környezetet és felfedezze a játékban rejlő lehetőségeket.

A pixelart stílus egyedi és könnyen felismerhető vizuális azonosítást ad a játéknak. A karakterek megjelenése és tulajdonságai befolyásolják a játékosok érzelmi kötődését és azonosulását velük, a pixelart stílus pedig egyedi esztétikai élményt nyújt, mely visszaidézi a retró játékok hangulatát és emlékeket ébreszt a múltból.

Ezen elemek együttesen teszik élvezetessé és motiválóvá a játékot. Az interakció, érdekes puzzle játékok, animációk, vizualizációk és a helyes karakter tervezés kiemelt szerepet játszanak a felhasználói élmény erősítésében és a játékosok motiválásában. Fontos hangsúlyozni, hogy a játékok interaktivitásával és érdekességével szorosan összefüggő, ezeknek az elemeknek a hiánya csökkentheti a játék vonzerőjét és nem motiválja a felhasználót a játék folytatására.

Kulcsszavak— puzzle játékok, animáció, vizualizáció, interakció, karakter tervezés, pixelart

Rezumat

Printre elementele care contribuie la experiența utilizatorului și la motivația jocurilor se numără jocurile de tip puzzle, animațiile, vizualizările, interacțiunile și designul propriu-zis al personajelor. Aceste componente importante nu numai că fac jocul atractiv din punct de vedere estetic, dar asigură și participarea activă și plăcerea jucătorului.

Jocurile puzzle sunt provocări interesante care necesită gândire creativă și abilități de rezolvare a problemelor pentru a le rezolva. Sarcinile variate încurajează jucătorul să facă pași logici, combinând și potrivind diferite elemente sau obiecte pentru a ajunge la o soluție.

Animațiile și vizualizările adaugă dinamism și viuiciune jocului. Aceste mișcări, tranziții și efecte spectaculoase fac ca lumea jocului să fie mai realistă și mai captivantă, capturând atenția jucătorului. Animațiile ajută la menținerea în mișcare a personajelor și a mediului înconjurător, sporind interactivitatea și plăcerea jocului.

Interacțiunea este unul dintre cele mai importante elemente ale experienței de joc, deoarece permite jucătorului să participe activ la lumea jocului. Acțiunile și alegerile jucătorului influențează jocul și povestea. Interacțiunea îi oferă jucătorului posibilitatea de a controla personajul, de a manipula mediul și de a explora posibilitățile jocului.

Stilul pixel art conferă jocului o identitate vizuală unică și ușor de recunoscut. Aspectul și atributele personajelor influențează conexiunea emoțională și identificarea jucătorilor cu acestea, iar stilul pixel art oferă o experiență estetică unică, care evocă atmosfera jocurilor retro și aduce aminte de trecut.

Aceste elemente se combină pentru a face jocul plăcut și motivant. Interacțiunea, jocurile de puzzle interesante, animațiile, vizualizările și un design bun al personajelor joacă un rol esențial în îmbunătățirea experienței utilizatorului și în motivarea jucătorilor. Este important de subliniat faptul că lipsa acestor elemente, care sunt strâns legate de interactivitatea și interesul jocurilor, poate reduce atractivitatea jocului și nu îl motivează pe utilizator să continue să joace.

Cuvinte cheie— jocuri de puzzle, animație, vizualizare, interacțiune, design de personaje, pixel art

Abstract

Elements that contribute to the user experience and motivation of games include puzzle games, animations, visualizations, interactions and character design itself. These important components not only make the game aesthetically appealing, but also ensure active participation and enjoyment for the player.

Puzzle games are exciting challenges that require creative thinking and problem-solving skills to solve. The varied tasks encourage the player to take logical steps, combining and matching different elements or objects to reach a solution.

Animations and visualizations add dynamism and liveliness to the game. These movements, transitions and spectacular effects make the game world more realistic and immersive, capturing the player's attention. Animations help to keep the characters and their environment in motion, adding to the interactivity and enjoyment of the game.

Interaction is one of the most important elements of the game experience, as it allows the player to actively participate in the game world. The player's actions and choices affect the gameplay and story. Interaction gives the player the opportunity to control the character, manipulate the environment and explore the possibilities of the game.

The pixel art style gives the game a unique and easily recognizable visual identity. The appearance and attributes of characters influence the emotional connection and identification of players with them, and the pixel art style provides a unique aesthetic experience that evokes the atmosphere of retro games and brings back memories of the past.

These elements combine to make the game enjoyable and motivating. Interaction, interesting puzzle games, animations, visualizations and good character design play a key role in enhancing the user experience and motivating players. It is important to emphasize that the lack of these elements, which are closely linked to the interactivity and interest of the games, can reduce the attractiveness of the game and do not motivate the user to continue playing.

Keywords— puzzle games, animation, visualization, interaction, character design, pixel art

Tartalomjegyzék

1. Bevezető	10
2. Játékok régen és most	11
2.1. A videójátékok fejlődése	11
2.2. 2D és 3D játékok	11
3. Felhasznált technológiák	13
3.1. A Unity játékmotor	13
3.2. Felhasználói felület (UI)	14
3.3. A Unity alapjai	17
3.4. A C# programozási nyelv	19
4. Algorythmic Universe	22
4.1. Algorythmic Universe felépítése	22
4.2. A kinézet megalapozása	23
4.3. A karakter megalkotása	23
4.4. A karakter életre keltése	23
4.5. A puzzle elemek összeállítása	30
4.6. A tipprendszer olyan, mint egy rejtett oktató	32
4.7. A tipprendszer megalkotása	34
Összefoglaló	38
Ábrák jegyzéke	39
Irodalomjegyzék	40

1. fejezet

Bevezető

Az évtizedek során a videójátékok népszerűsége óriási növekedést mutatott, ami egy virágzó játékiparhoz vezetett. Az emberek különféle okokból kezdenek el játszani, ideértve a kapcsolódást, a versenyképes készségek fejlesztését, a szórakoztatást és még oktatási célokat is. Következésképpen a játékipar a világ egyik legnagyobb és leglenyűgözőbb üzletévé vált. Az indie játékcégek és a nagyvállalatok minden nap új játékokat mutatnak be a játékosoknak, miközben a megjelenési dátumokkal kapcsolatos hírek elárasztják a közösségi médiát és a hirdetéseket.

A játékfejlesztés és -tervezés iránti növekvő érdeklődés számos programozót és fejlesztőt vonzott. Sok törekvő egyén azonban ijesztőnek találja a játékfejlesztést a szükséges készségek és ismeretek széles skálája miatt. A programozástól és az animációtól a hangtervezésig és a környezetművészetig ezeknek a tudományágaknak az alapvető ismerete elengedhetetlen a játékok létrehozásához. Bár a nagy csapatok és játéktúdiók, mint például a Ubisoft, a Naughty Dog, a Square Enix és a CD Projekt Red általában AAA minőségű játékokat fejlesztenek, ez nem jelenti azt, hogy az egyének nem képesek önállóan figyelemreméltó játékokat létrehozni. Valójában számos indie játékfejlesztő alkotott meghökkentő játékokat, mint például Markus "Notch" Persson Minecraftja, Eric Barone Stardew Valleyje és Toby Fox Undertaleje.

Szakdolgozatomban egy átfogó útmutatót kívánok bemutatni egy puzzle játék létrehozásának folyamatáról. A játékfejlesztéshez szükséges alapvető készségek megismertetésére koncentrálok, felhasználva a széles körben használt C# programozási nyelvet és a népszerű Unity játékmotort. Konkrétan a Unity játék létrehozásának 2D aspektusára fogok koncentrálni, mivel a puzzle játékomat ebben a formátumban terveztem. Dolgozatom konklúziójaként az a célom, hogy a programozókat és fejlesztőket szilárd alapokkal ruházzam fel a kirakós játékok fejlesztésében, lehetővé téve számukra, hogy önállóan készítsék el saját játékaikat.

2. fejezet

Játékok régen és most

2.1. A videójátékok fejlődése

Alan Turing, a számítástechnika és a mesterséges intelligencia úttörője, akaratlanul is megalkotta az első számítógépes játékot 1947-ben – egy számítógépes sakkprogramot [Tur95]. Kezdetben a cél a mesterséges intelligencia tesztelése volt, nem pedig az emberek számára játszható játék létrehozása. 1952-ben Turing egy kollégájával együtt tesztelte a programot, magára vállalva a számítógép szerepét. Ez az úttörő játék számos matematikust és informatikust inspirált munkája folytatására.

A Pong 1972-es megjelenése jelentős mérföldkövet jelentett a játéktörténelemben [GLBC20]. Ez az arcade játék rendkívül népszerűvé vált, olyannyira, hogy az arcade gépek gyakran elakadtak a behelyezett érmék puszta száma miatt.

1980 júliusában Toru Iwatani Pac-Man ötletét egy hiányzó pizzaszelet váltotta ki. Ez a játék példátlan sikert ért el, mint az első arcade sláger, amelyet otthon is lehetett játszani konzollal.

Gyorsan előre a 21. század elejére, szemtanúi vagyunk a dedikált hírcsatornák, streaming szolgáltatások, online játékboltok és a népszerű játékokon alapuló áruk tömkelegének. Olyan cégek, mint az Electronic Arts, a Ubisoft és a CD Projekt Red elkötelezték magukat a játékfejlesztés mellett. Kétségtelen, hogy a játékipar az egyik leggyorsabban növekvő és legjövedelmezőbb iparággá vált világszerte.

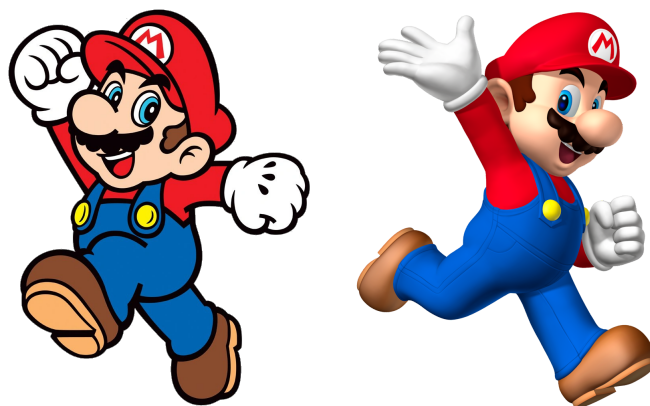
2.2. 2D és 3D játékok

A számítógépes játékok kezdeti korszakában a technológia korlátai a kétdimenziós (2D) természetére korlátozták őket. A technológia fejlődése azonban mára lehetőségeket nyitott háromdimenziós (3D) szimulációk létrehozására megfelelő eszközök segítségével. A 3D nyomtatás megjelenése tovább erősíti ezeket a képességeket, lehetővé téve számunkra, hogy olyan forgatókönyveket tapasztaljunk meg, amelyek jellemzően elérhetetlenek vagy túl kockázatosak lennének a valóságban. A 3D-s szimulációk a vezetés és a repülés elsajátításától a Naprendszer szimulálásáig a megszokott hatókörünkön túl is magával ragadó élményeket kínálnak.

Most pedig nézzük meg a 2D és 3D játékok definícióit:

- 2D játékok: Ezeket a játékokat kétdimenziós térben tervezték, jellemzően lapos grafikával és oldalsó vagy felülről lefelé görgető perspektívával. A játékmenet és a látvány síkon zajlik, mélység vagy háromdimenziós tér érzékelése nélkül.
- 3D-s játékok: Ezzel szemben a 3D-s játékok háromdimenziós grafikát és környezetet használnak, ami a mélység és a valóság érzetét kínálja. A játékmenet és a látványvilág egy

virtuális térben épül fel, így magával ragadóbb és vizuálisan vonzóbb élményt nyújt. A tárgyak, karakterek és környezetek mélységgel renderelhetők, élethűbb és interaktívabb játékvilágot teremtvé.



2.1. ábra. 2D illetve 3D karakter hasonlítás

A 2D-s játékokban a játék világa két dimenzióban létezik, nevezetesen az X és Y tengelyen. Ezek a játékok lapos grafikát, úgynevezett sprite-okat használnak, és nincs háromdimenziós geometriájuk. A vizuális elemek lapos képként jelennek meg a képernyőn, és a kamera, jellemzően egy ortografikus kamera, nem rendelkezik perspektívával. A 2D-s játékokban a játékosok mozgása a vizuálisan lapos környezet miatt általában az X-Y tengely mentén balra, jobbra, felfelé és lefelé korlátozódik. A népszerű 2D-s játékok közé tartozik például a Super Mario Bros, a Pac-Man és a Donkey Kong.

A 2D-s játékok mozgáskorlátozottsága azonban nem jelenti azt, hogy a 3D-s játékokhoz képest kevésbé élvezhetőek. Az élvezet az egyéni preferenciáktól függ. Továbbá az alkalmi és könnyű játékelményt kereső játékosok számára a 2D-s játékok gyakran megfelelőbbek, mivel a modern 3D-s játékok általában igényesebbek és nagyobb tárhelyet foglalnak.

Másrészt a 3D-s játékok magával ragadóbb élményt nyújtanak. Ezek a játékok háromdimenziós geometriát használnak, ahol a játéktárgyak, a környezet és a karakterek mélységgel és térfogattal renderelődnek. A játéktárgyak felületére anyagokat és textúrákat alkalmaznak, hogy szilárd és valósághű látványt hozzanak létre. A 3D-s játékokban a játékosok a valós világot szimulálva három dimenzióban navigálhatnak és fedezhetik fel a játékvilágot. A mozgás több irányban lehetséges az X, Y és Z tengely mentén, többek között felfelé, lefelé, balra, jobbra, előre és hátra. A 3D-s játékok a 2D-s játékokhoz képest gyakran szélesebb körű lehetőségeket és célokat kínálnak.

A 2D-s játékokban a karakterek mozgása általában két dimenzióra korlátozódik, így könnyebben irányítható. Ezzel szemben a 3D-s játékok nagyobb kihívást jelentenek a játékosok számára, mivel alkalmazkodniuk kell a háromdimenziós mozgáshoz. Ehhez olyan billentyűket használhatnak, mint a WASD a mozgáshoz, a szóköz az ugráshoz, és az egér a nézelődéshez.

Azért koncentrálok írásomban a 2D-s játékfejlesztésre, mert ez szolgál a legalapvetőbb és legkönnyebben megközelíthető belépési pontként a játékfejlesztés elsajátításához. A 2D-s játékfejlesztés elsajátításával az olvasók megszerezhetik a szükséges tudást és alapokat ahhoz, hogy zökkenőmentesen át tudjanak térni a 3D-s játékfejlesztésre, ha úgy döntenek, hogy ezt teszik.

3. fejezet

Felhasznált technológiák

A felhasznált technológiák közül, előszóként, kiemelném a játékmotort. A játékmotor egy speciális szoftver, amelyet kifejezetten videojátékok készítésére terveztek. Olyan platformként szolgál, amely lehetővé teszi a fejlesztők számára, hogy különböző platformokra, például számítógépekre, konzolokra és mobiltelefonokra készítsenek játékokat. A játékmotor elsődleges célja, hogy könnyedén megoldja a játékkal kapcsolatos általános feladatokat, például a renderelés, a fizika és a bevitel, amelyek platformfüggetlenek lehetnek. E feladatok kezelésével a fejlesztők, a művészek, a tervezők és programozók azokra az egyedi szempontokra összpontosíthatnak, amelyek kiemelik játékukat.

A játékmotorok kulcsfontosságú összetevői jellemzően tartalmazzák egy renderelőt a 2D és 3D tervezéshez, egy fizikamotort, szkriptfunkciókat, hangkiegészítőket és animációs eszközöket. Sok játékmotor olyan újra felhasználható komponenseket is biztosít, mint a karaktermodellek, az alapvető hangok és a sablonvilágok. Ez a sokoldalúság lehetővé teszi, hogy a játékmotorokat a játékfejlesztésen túl más célokra is lehessen használni, például szoftvertervezésre, 2D és 3D animációk készítésére, sőt még zenei produkciókra is. Az animációhoz és a zenéhez azonban léteznek speciális eszközök, illetve mivel azokat kifejezetten ezekre a területekre fejlesztették ki, az ezeken a területeken dolgozó szakemberek gyakran előnyben részesítik a speciális igényeik kielégítésére kifejlesztett szoftverek használatát. Ezért, míg a játéktervezési szoftverek vagy játékmotorok különböző elemeket tudnak kezelni, a szakemberek olyan speciális eszközöket választhatnak, amelyeket kifejezetten az adott szakterületre szabtak.

3.1. A Unity játékmotor

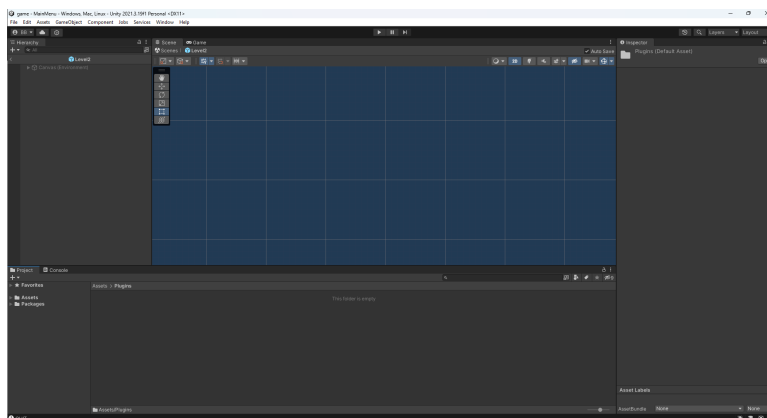
2005 óta a Unity világszerte az indie játékfejlesztők kedvelt választása lett. Nemcsak 2D-s és 3D-s játékok fejlesztésére egyaránt alkalmas, hanem a virtuális valóság (VR) és a kiterjesztett valóság (AR) tervezésében is kiválóan teljesít. A motor évente frissítéseken esik át, új funkciókat és tartalmakat vezetve be. A Unity emellett virágzó közösségnek is örvend, és hatalmas eszköztárral büszkélkedhet, amely ingyenes és fizetős eszközök széles választékát kínálja. A Unity másik jelentős előnye a hozzáférhetősége, mivel ingyenesen elérhető, így bárki könnyedén elkezdheti a játékfejlesztés elsajátítását. [Haa14]

Néhány népszerű játék, amelyet a Unity segítségével fejlesztettek, például a Cuphead, az Inside és az Ori and the Blind Forest.

Az ok, amiért a szakdolgozatomhoz a Unity Engine-t választottam, az az egyedül végzett játékfejlesztésre való alkalmassága. Jellemzően a játékfejlesztés csapatmunkát igényel az érintett komponensek sokasága miatt. Az olyan készségek elsajátítása, mint az animáció, a hangtervezés, a környezettervezés és a kódolás időigényes vagy meghaladja egy egyén képességeit. A

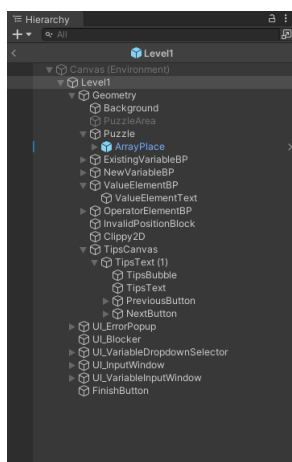
Unity azonban bizonyos komponenseket ingyenesen biztosít, így a fejlesztők a programozásra koncentrálhatnak, és egyszerűsíthetik a fejlesztési folyamatot. Egyik figyelemre méltó funkciója az eszköztár, amely hozzáférést biztosít a játékkész eszközök hatalmas gyűjteményéhez, amelyeket saját játékaikban felhasználhatunk. Akár egyénileg, akár csapatban dolgozunk, a Unity hatékony választásnak bizonyul. Ebben a fejezetben mélyebben elmélyedünk a Unity Engine-ben, feltárva annak különböző jellemzőit és funkcióit.

3.2. Felhasználói felület (UI)



3.1. ábra. Unity felhasználói felület

A kezdők számára a Unity felhasználói felülete összetettsége miatt kezdetben túlterhelőnek tűnhet. Időbe telhet, amíg megszokják a különböző komponensek közötti navigációt, viszont miután elsajátítják, nagyon egyszerű és intuitív alkotni vele. Ebben a szakaszban végig vezetem a felület elrendezésén, kezdve az alapértelmezett elrendezéssel, amely egy új projekt létrehozásakor jelenik meg.



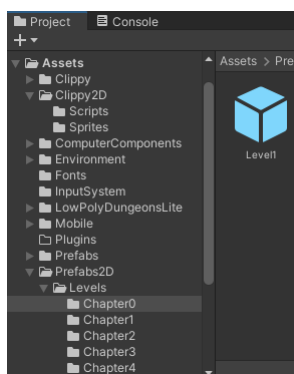
3.2. ábra. Hierarchia ablak

A bal felső sarokban található a Hierarchia ablak, amely a jeleneten belüli összes játékbjektumot megjeleníti. A jeleneten belül végrehajtott bármilyen elem hozzáadása vagy eltávolítása megjelenik a Hierarchia ablakban.

Unityben a játékobjektumok alapvető entitásokként szolgálnak, amelyek karaktereket, tulajdonságokat és környezeteket képviselnek. Tárolóként működnek a komponensek számára, amelyek funkcionalitást biztosítanak számukra. Egy üres játékobjektum önmagában nem hajt végre semmilyen konkrét műveletet, de ha egy komponenst csatolunk hozzá, akkor konkrét képességeket kap. Például, ha egy üres játékobjektumhoz kattintási komponenst adunk hozzá, akkor a játékos képes lesz rákattintani az objektumra, amely erre reagálni fog.

Mivel egy puzzle rengeteg elemet tartalmaz, fontos volt, hogy logikusan legyen minden rendezve a Hierarchia ablakban, különben nagyon hamar kezelhetetlenné váltak a játékobjektumok.

Unityben a jelenetek magukba foglalják a játékunkat alkotó objektumokat. Minden egyes jelenetre úgy is gondolhatunk, mint egy különálló szintre a játékban belül. Ezen kívül a jeleneteket használhatjuk a főmenü létrehozására, ahol a játékosok elindíthatják a játékot, testre szabhatják a beállításokat, és kiléphetnek a játékból. Több jelenet kombinálható, hogy teljes játéklélményt teremtsen.

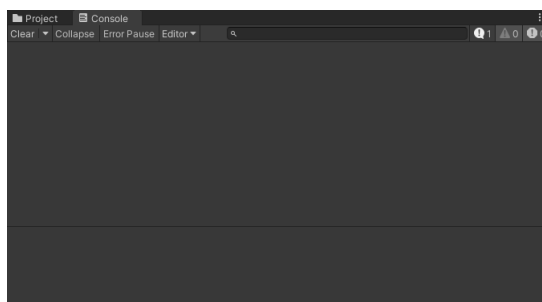


3.3. ábra. Projekt ablak

A Hierarchia ablak alatt található a Projekt ablak. Ez az ablak több célt is szolgál. Először is, megjeleníti az aktuális projekthez tartozó összes fájlt, áttekintést nyújtva annak tartalmáról. Másodszor, a projekten belüli navigáció elsődleges eszköze, amely lehetővé teszi a felhasználók számára a fájlok elérését és kezelését.

Akárcsak a Hierarchia ablaknál, itt is elengedhetetlen, hogy fájljainkat a lehető legrendezettebben tároljuk, ezzel jelentősen megkönnyítve, és felgyorsítva az alkotásaink létrejöttét.

Ezenkívül a Projekt ablak lehetővé teszi az új fájlok létrehozását a projekthez, megkönnyítve a projekt erőforrásainak bővítését és szervezését.

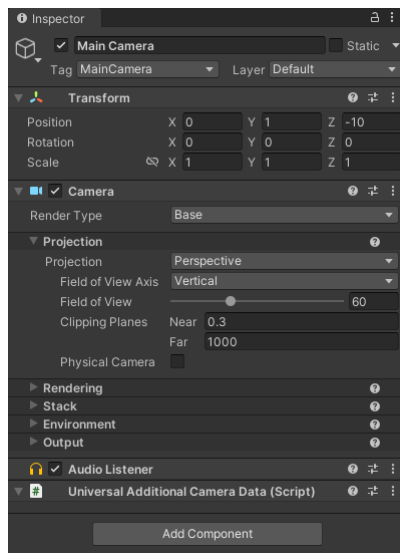


3.4. ábra. Konzol ablak

A Projekt ablak mellett található a Konzol ablak, amely nagy jelentőséggel bír. Ez az ablak döntő szerepet játszik, mivel megjeleníti a projektünkkel kapcsolatos figyelmeztetéseket,

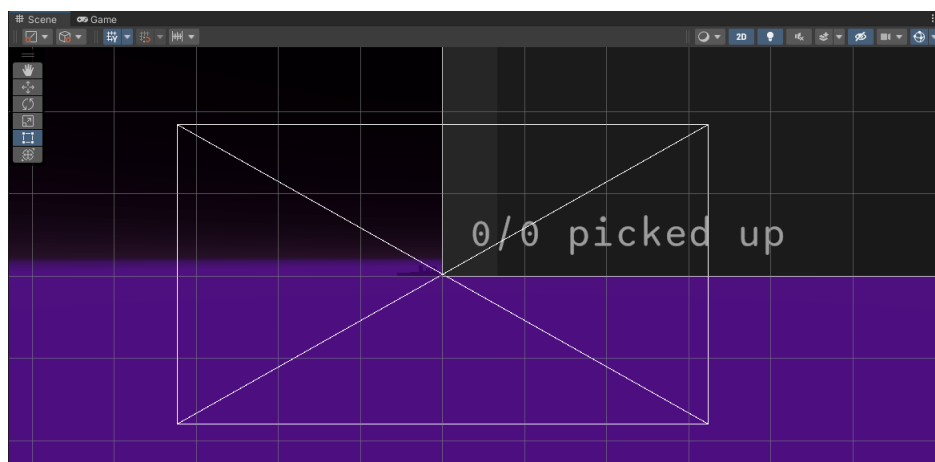
hibákat és egyéb fontos üzeneteket. Ezek az üzenetek elengedhetetlenek játékunk zökkenőmentes működésének biztosításához, és a Konzol ablak értékes eszközként szolgál a hibák hatékony azonosításához és kijavításához.

Rengeteg fejfájást előzött meg, és rengeteg hibát sikerült kiküszöbölni a hasznos Log-ok használatával.



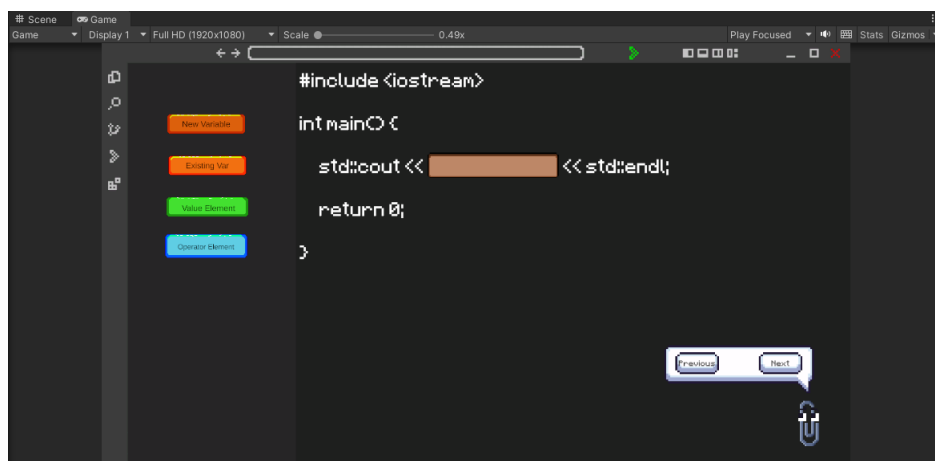
3.5. ábra. Inspector ablak

A felhasználói felület jobb oldalán található az Inspector ablak. Ez az ablak részletes képet nyújt az aktuálisan kiválasztott játékbjektumhoz tartozó komponensekről. Lehetővé teszi számunkra, hogy hozzáadjuk, eltávolítsuk és testre szabjuk ezeket a komponenseket a projektünk egyedi követelményeinek megfelelően. Az Inspector ablak kényelmes eszközként szolgál a játékbjektumok tulajdonságainak és funkcióinak kezeléséhez és módosításához a Unityben. Talán az egyik legfontosabb szerepe volt a játék fejlesztésében az Inspector ablaknak. Mivel rengeteg specifikus és különböző játékelemekből alakult ki a puzzle játék, a megszokottnál több komponensre volt szükség egy-egy elemnél. Időm legtöbb részét az Inspector menüben töltöttem el, mivel rengeteg finomhangolást igényeltek a játékelemek, hogy azok megfeleljenek a saját standardjaimnak.



3.6. ábra. Scene ablak

A felhasználói felület közepén elhelyezett Scene ablak interaktív nézetet nyújt a jelenetünkben felépített virtuális világról. Lehetővé teszi számunkra, hogy vizuálisan interakcióba lépjünk a különböző komponensekkel, például a díszlettel, a karakterekkel, a platformokkal és a környezetekkel. A Scene ablak a játékbjektumaink vizuális megjelenítésére szolgál, lehetővé téve számunkra, hogy navigáljunk és manipuláljuk őket a kívánt játékvilágunk megtervezése és felépítése érdekében. Elengedhetetlen volt a Scene ablak a játék kinézetének megalkotásában. Rengeteg fázison esett át a játék felhasználói felülete, egyértelműleg létfontosságú volt, hogy könnyen tudjam mozgatni, illetve méretezni az elemeket, hogy lássam, mennyire alkotnak egy együttes látványt.

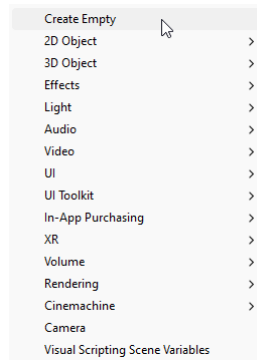


3.7. ábra. Játék ablak

A Jelenet ablak mellett található a Játék ablak, amely vizuálisan ábrázolja, hogy mit lát a játékos a játék során. Megjeleníti a játékot úgy, ahogyan az a játékos számára megjelenne, lehetővé téve számunkra, hogy előzetesen megnézzük és értékeljük a játékunk vizuális elemeit, animációit és általános élményét. A Jelenet ablakkal ellentétben a Játék ablak elsősorban vizuális célokat szolgál, és általában nem támogat interaktív funkciókat.

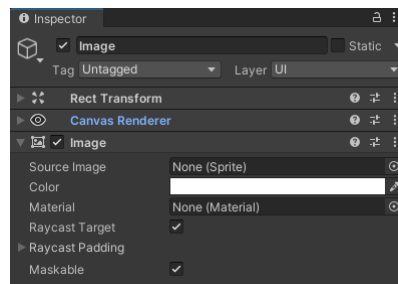
3.3. A Unity alapjai

Ebben a részben néhány alapvető funkciót tekintünk át, amelyeket a Unityvel elvégezhetünk. A következő fejezetben csak azokat a funkciókat mutatom be, amelyeket rendszeresen használunk. Az itt bemutatott lépések mindegyikét többféleképpen is megtehetjük, így az olvasók felfedezhetik a felhasználói felületet, és megtalálhatják saját preferenciáikat.



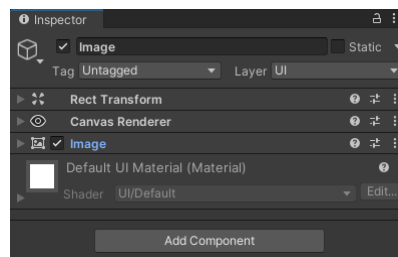
3.8. ábra. Objektum létrehozása

Egy üres GameObject létrehozásához egyszerűen kattintson a jobb gombbal a hierarchiára, és válassza a "Create Empty" lehetőséget a kontextusmenüből. Mint korábban említettük, egy üres GameObject önmagában nem rendelkezik semmilyen funkcióval; komponenseket kell hozzáadnunk. A mellékelt képen különböző típusú objektumok, például 3D objektumok, 2D objektumok és UI objektumok létrehozásának lehetőségei is láthatók. Ezekhez az objektumokhoz már előre létező komponenseket csatoltak meghatározott célokra. Például, ha a "UI"-ra kattintunk, és az "Image"-et választjuk, akkor egy "New Image" nevű GameObject jön létre, amelyhez már hozzá van adva egy "Image" komponens, ami képek betöltésére szolgál.



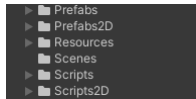
3.9. ábra. Kép komponens

Ha komponenst szeretne hozzáadni egy GameObjecthez, válassza ki a GameObjectet a Hierarchia ablakban, majd kattintson az "Add Component" gombra az Inspector ablakban. Ezután keresse meg a kívánt komponenst, és adja hozzá a GameObjecthez.



3.10. ábra. Komponens hozzáadása

A projektünkben sok fájlt fogunk létrehozni, ezért fontos, hogy a fájlokat mappákba rendezzük. Az embereknek különböző preferenciái vannak a projektjeik szervezésével kapcsolatban, de én azt találtam hatékonynak, hogy fájltípusok szerint szervezzük.



3.11. ábra. Projekt rendezettsége

Amint a fenti példában látható, a fájlok típusuk szerinti rendszerezése, például a puzzle pályák elhelyezése a "Levels" mappában, jobb kezelést és a meghatározott típusú fájlok könnyű elérését teszi lehetővé. Az egyes kategóriákon belül almappák hozhatók létre az egyes fájltypusok alapján történő további szervezéshez. Ez a megközelítés segít a létrehozott fájlok áttekinthetőségének fenntartásában és megkönnyíti a hatékony fájlkeresést.



3.12. ábra. Eszközsáv

A fenti eszköztárat használjuk a játékobjektumunk konfigurálására a Scene-ben. Ablakban:

- Hand tool: ezzel az eszközzel pásztázhatunk a jelenetben.
- Move Tool: a kiválasztott GameObject mozgatása.
- Rotate Tool: a kiválasztott GameObject forgatása.
- Scale Tool: a GameObject átméretezése minden tengelyen egyszerre.
- Rect Tool: téglalap GameObject méretezése, pozicionálása, mérete és rögzítése.

Ezek a Unity alapvető funkciói, és számos további funkció és megközelítés áll rendelkezésre a játékfejlesztéssel kapcsolatos egyéni preferenciáink alapján. Egy későbbi fejezetben az első 2D-s puzzle játékunk elkészítése során további Unity-funkciókba fogok belemerülni, mivel ezeket könnyebb elmagyarázni és felfedezni a gyakorlati alkalmazás kontextusában.

3.4. A C# programozási nyelv

A C#, amely az F# különböző jellemzőiből merített ihletet, egy programozási nyelv, amelyet 2000-ben mutattak be a nyilvánosságnak. A Windows platformon történő alkalmazásfejlesztésre tervezett sokoldalú nyelv, működéséhez a .NET keretrendszerre van szükség. A C# a programozási feladatok széles körére alkalmas, és különösen kedvelt Windows-alkalmazások és játékok fejlesztéséhez. Emellett egyes webfejlesztők a webes alkalmazások készítésekor is a C# nyelvet választják, és a mobilfejlesztés területén is egyre népszerűbb.

Bár a C# viszonylag könnyen tanulható és olvasható, rugalmassága miatt kihívást jelenthet a teljes elsajátítása. Ez egy összetett nyelv, amelynek elsajátítása több időt igényelhet az egyszerűbb nyelvekhez, például a Pythonhoz vagy a HTML-hez képest. A felhasználóknak jelentős mennyiségű kódolási tudást kell elsajátítaniuk ahhoz, hogy fejlett programokat hozhassanak létre. A C# megtanulásába fektetett erőfeszítés azonban megéri. A C# nyelv a munkaerőpiacon igen keresett készség, számos technológia-központú vállalat aktívan keresi az erős C# szak-tudással és tapasztalattal rendelkező fejlesztőket. Következésképpen a C#-tudással rendelkező személyek kiváló álláslehetőségeket élveznek.

A szakdolgozatomhoz a C# nyelvet választottam a sokoldalúsága miatt. Az olvasók nemcsak a játékfejlesztéshez tanulhatják meg a C# nyelvet, hanem az IT-ipar különböző területein is növelhetik kilátásaikat.

```

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CharacterController : MonoBehaviour
{

    // Start is called before the first frame update

    void Start()

    {

    }

    // Update is called once per frame

    void Update()

    {

    }

}

```

3.1. kódrészlet. C# példakód

A mellékelt példa egy C# szkriptet szemléltet, és kiemeli annak strukturált jellegét, amely viszonylag könnyen érthető. A C# nyelvben egy osztály egy objektum tervrajzaként szolgál, amely hasonlít a valós objektumokhoz, amelyek tulajdonságokkal és funkciókkal rendelkeznek. Egy osztályon belül metódusok jönnek létre az objektumok funkcióinak és tulajdonságainak meghatározására. A példában szereplő "CharacterController" osztály olyan metódusokat tartalmaz, mint a "Start" és a "Update", amelyek meghatározott funkciói az előző zöld kommentársorokban vannak megadva. További metódusok hozhatók létre megfelelő nevekkal, például "CharacterAnimation", "CharacterDialogue" vagy "CharacterIdle". A metódusnevek nem tartalmazhatnak szóközöket, és tartalmukat szögletes zárójelek közé kell zárni.

A megjegyzések a C# nyelvben kizárólag a kód magyarázatára és olvashatóságára szolgálnak, és nem tartalmaznak semmilyen funkciót magában a kódban. Az egysoros megjegyzések kettős `/*` előremenő kötőjellel kezdődnek. A szkriptben szereplő `using` utasítások névterekre vonatkoznak, amelyek lehetővé teszik az adott névtéren belüli metódusok hatékony hivatkozását. Segítenek csökkenteni a névtér ismételt megírásának szükségességét a metódusok használatakor. Például a `using System;` névtér alkalmazásával a kód a `System.Console.WriteLine("Hello World!");` helyett `Console.WriteLine("Hello World!");` lehet.

Ezek a magyarázatok a C# Unityben való felhasználásának alapvető szempontjait fedik le. A C# átfogó megértése meghaladja e szakdolgozat kereteit.

4. fejezet

Algorythmic Universe

Ebben a fejezetben egy 2D-s puzzle játék kigondolásával és elkészítésével foglalkozunk. Ezután a fejezet után rendelkezni fogunk azokkal a készségekkel és ismeretekkel, amelyekkel egy ténylegesen működő játékot készíthetünk és megkezdjük utazásunkat a játékfejlesztés felé. Egy 2D puzzle játékot fogunk készíteni "Algorythmic Universe" címmel.

4.1. Algorythmic Universe felépítése

Mielőtt belevetnénk magunkat a játékunk fejlesztésébe, szeretném, ha átgondolnánk, hogyan fogjuk elkészíteni a játékot: mi a téma, mit szeretnénk, hogy a játékos mit tapasztaljon, hogyan emeljük ki a játékunkat a többi játék közül, milyen játékot készítünk, és még sok más kérdés, ami eszünkbe juthat. Ha ezeket a kérdéseket magunknak tesszük fel, az nem csak abban segít, hogy fejben alakítsuk ki a játékot, hanem arra is ösztönöz, hogy arra koncentráljunk, hogy a játékot úgy készítsük el, ahogyan azt mi elképzeltük. Tapasztalatom szerint, amikor egy új játékot készítek, hajlamos vagyok több funkciót létrehozni hozzá, mint amennyit eredetileg terveztem. Ez nem rossz szokás, a játék lehet, hogy jobb lesz, de néha próbáljunk meg nem elfeledkezni, hogy megkérdezzük magadtól: "Ezt akarom, hogy a játékos ezt tapasztalja?". Például egy olyan játékot készítünk, ami a különböző kódok kiegészítéséről szól, de egy pálya létrehozásakor túl sok részletet hagyunk ki, túl komplikált feladványokat építünk a játékunkba, akkor a játékos lehet, hogy elveszettnek fogja érezni magát. Ilyenkor vissza kell tekintenünk és meg kell kérdeznünk magunktól, hogy "mi van a játék tanítási jellegével?". Például az "Algorythmic Universe" című játékban a játékosunk egy hiányos kódrészlettel találkozik. Mivel a játék fő célja a tanítás [May19], apró tippekkel kell a játékosunkat a helyes megoldás felé vezetni, ezt a feladatot tölti be a játékba implementált tipprendszer.

Összefoglalva, a játékunk meghatározása:

- Játékos élmény: barátságos és szórakoztató tanulás
- Alapmechanika: 3D platformer és 2D puzzle
- Téma: tanuló játék

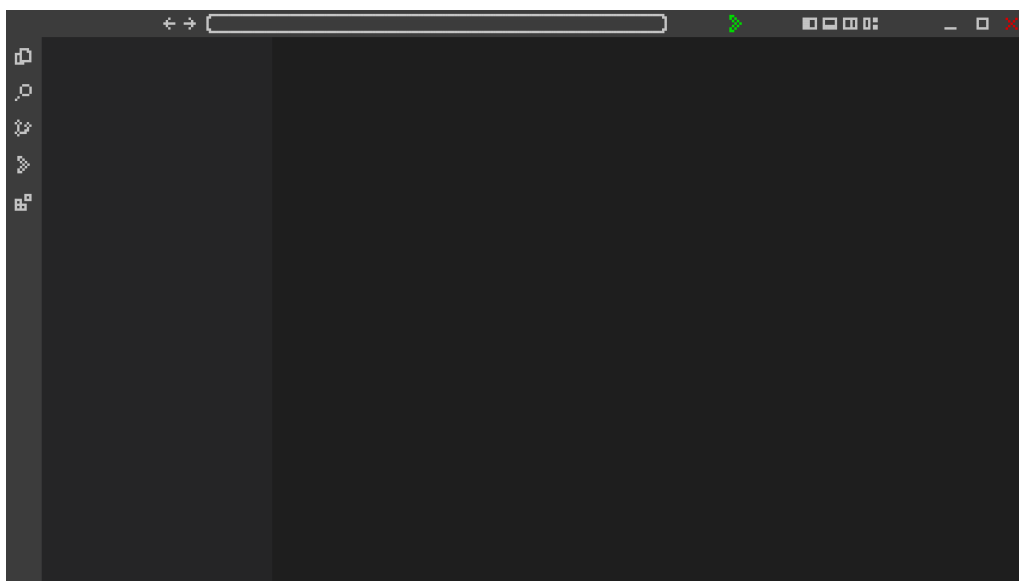
Mivel meghatároztuk játékunkat, és tudjuk, hogy melyik részén kell mi dolgozzunk, készen is állunk a játékunk elkészítésére!

4.2. A kinézet megalapozása

A játékfejlesztés egyik kezdeti feladata a játékkörnyezet létrehozása. Elkészíthetjük saját kezűleg is a teljes környezetet, viszont ez a folyamat egyszerűsíthető játékkész eszközök felhasználásával.

Eme játék fejlesztése közben arra gondoltam, hogy retró stílusban készítem el a környezetet, főként pixelart stílusú elemeket használva.

A puzzle játékunk háttere a Visual Studio Code fejlesztési környezetből merített ihletet, így elkészítettem ennek a pixeles változatát. Úgy gondoltam ez a felület lesz a legjobb a játékunkban, mivel aki programozni szeretne, legtöbbször ezzel a felülettel fog találkozni, ami a Visual Studio Code sokoldalúságából és flexibilitásából adódik. Mivel emellett rengeteg kiterjesztés is készül rá különböző fejlesztőktől, ha valaki nagyon szeretné, minden fejlesztési felületet lecserélhet, és mindent meg tud oldani a VS Code segítségével.



4.1. ábra. Visual Studio Code pixelart változata

4.3. A karakter megalkotása

A mi játékunkban egy karakter szerepel, a játékos segítőkész asszisztense, Clippy, aki játékunk során tippekkel és útbaigazítással látja el játékosunkat. A játékos az egér segítségével tud interakcióba lépni Clippy-vel, illetve magával a játékkal is.

Az ötlet, hogy Clippy, egy gémkapocs legyen a játékunk főarca, már a kezdetleges fázisban is megvolt. A fejlesztés legkorábbi szakaszában már eldőlt, hogyan is fog kinézni a karakterünk, mivel könnyen felismerhető, barátságos a dizájnja, illetve nagyon könnyen összeköthető a sablonkitöltéssel is, a világhírű *“It looks like you’re trying to write a letter. Would you like help with that?”* mondata. [BSPW19]

4.4. A karakter életre keltése

Véleményem szerint az animáció elengedhetetlen része egy minőségi játéknak. Fontos, hogy a játékos érezze, hogy a játék nem csak egy konstans kép, hanem van élet benne. Mi sem egyszerűbb az életet sugározni, mint megeleveníteni a játék béli karaktereket.

Ebben a részben Clippy animációinak létrehozására összpontosítunk.

A karakteranimációk létrehozására két elsődleges módszer létezik:

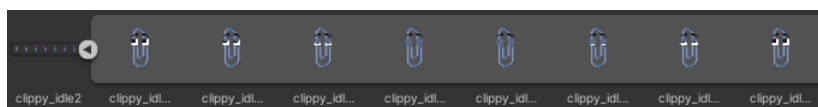
- Vázanimáció: Ez a megközelítés a karakter különböző részeinek összekapcsolását, valamint mozgásuk és interakcióik programozását foglalja magában. Hasonlít a csontszerkezethez, ahol az ízületek egymáshoz kapcsolódva alkotnak egy csontvázat.
- Sprite Sheet Animáció: Ez a technika egy objektum több képkockájából álló sprite sheet-et használ. Minden egyes képkocka egy enyhe mozgást ábrázol, és amikor ezeket a képkockákat gyors egymásutánban lejátszzák, a mozgás illúzióját keltik. Ez az animációs technika hasonlít a hagyományos rajzfilmanimációhoz.

Ehhez a projekthez a sprite sheet animációt választottuk módszerünknek, mivel a vázanimációhoz képest egyszerűbb, illetve az eredetileg elképzelt képünkhöz is közelebb áll. A karaktert a sprite-sheet nevű technológiával készítettem el, ami akárcsak a régi animált filmek, több képkockából áll, amelyek minimálisan térnek el egymástól, és gyors, egymás utáni sorrendbeni lejátszásuk eredményezik az animációt. (Lásd [4.2](#))

Egy sprite sheet létrehozása több lépést foglal magában, amelyeket az alábbiakban ismertetek:

- Sprite-ok megtervezése és létrehozása: Először el kell döntenünk, hogy hogyan is fog a karakterünk kinézni. Próbáljunk minél szimplisztikusabb dizájnt választani, hogyha retró stílusban dolgozunk, mivel a túl sok részlet elveszik a pixeles dizájnból. Sokféle grafikai tervezőprogram, illetve pixel art eszköz áll a rendelkezésünkre a sprite-ok elkészítéséhez. Minden sprite-ot egy átlátszó háttérben kell elhelyezni, hogy később könnyen kivethető legyen.
- Sprite-ok rendezése: Rendezzük az egyes sprite-okat logikus rácsmintázatba egy nagyobb vásznon belül. Ügyeljünk arra, hogy az egyes sprite-ok között egyenlő távolságot hagyjunk, hogy elkerülje a szomszédos sprite-ok átfedését vagy átlógását.
- Optimalizáljuk a sprite sheet-et: A sprite sheet véglegesítése előtt fontoljuk meg annak optimalizálását a fájl méret minimalizálása és a teljesítmény javítása érdekében. A technikák közé tartoznak:
 - A felesleges átlátszó területek eltávolítása az egyes sprite-ok körül, a fájl méret csökkentése érdekében.
 - Megbizonyosodunk, hogy a sprite-ok azonos méretűek, hogy később a felhasználásnál ne adódjon ebből probléma.
 - A redundáns vagy duplikált sprite-ok elkerülése, a felesleges adatok kiküszöbölése érdekében.
 - Veszteségmentes tömörítési algoritmus alkalmazása, például PNG-8 vagy PNG-24 formátum a fájl méret minőségromlás nélküli csökkentése érdekében.
- A sprite sheet exportálása: A sprite sheet-ek rendezése és optimalizálása után, exportáljuk azt egyetlen képfájlként. A sprite sheet-ek általános képfarmátumai közé tartozik a PNG, a GIF vagy a JPEG. Olyan képfarmátum választása, amely támogatja az átlátszóságot (például PNG), ha a sprite-ok átlátszó hátterűek.

A sprite sheet-ek használatával hatékonyan kezelhetünk és tölthetünk be több képet, csökkenthetjük a memóriahasználatot és optimalizálhatjuk a renderelési teljesítményt, ami népszerű technikává teszi őket a 2D-s játékfejlesztésben és animációban.

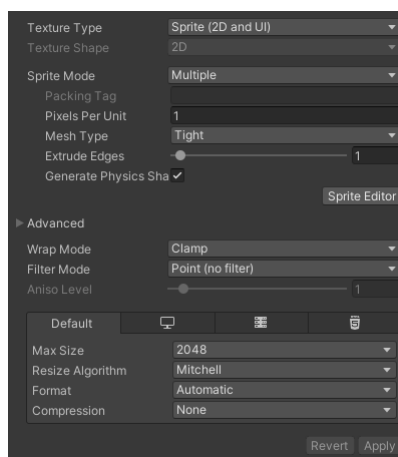


4.2. ábra. Clippy második idle animációjának a sprite sheet-je

Amikor megpróbálunk egy animációt a jelenetünkbe húzni, észrevehetjük, hogy a sprite nem jelenik meg. Ez a renderelési réteg problémái miatt van. Alapértelmezés szerint a karakter az alapértelmezett rétegre kerül, amely elsőként renderelődik, és amelyet a háttér és az előtér eltakarhat. Ahhoz, hogy a karakter láthatóvá váljon, az előtérrel azonos réteghez rendelhetjük. Alternatív megoldásként létrehozhatunk egy új réteget "Clippy" néven, és azt a "Background" réteg elé helyezhetjük. Ez a megközelítés javítja a skálázhatóságot, és segít nekünk a különböző elemek renderelésének megszervezésében és nyomon követésében.

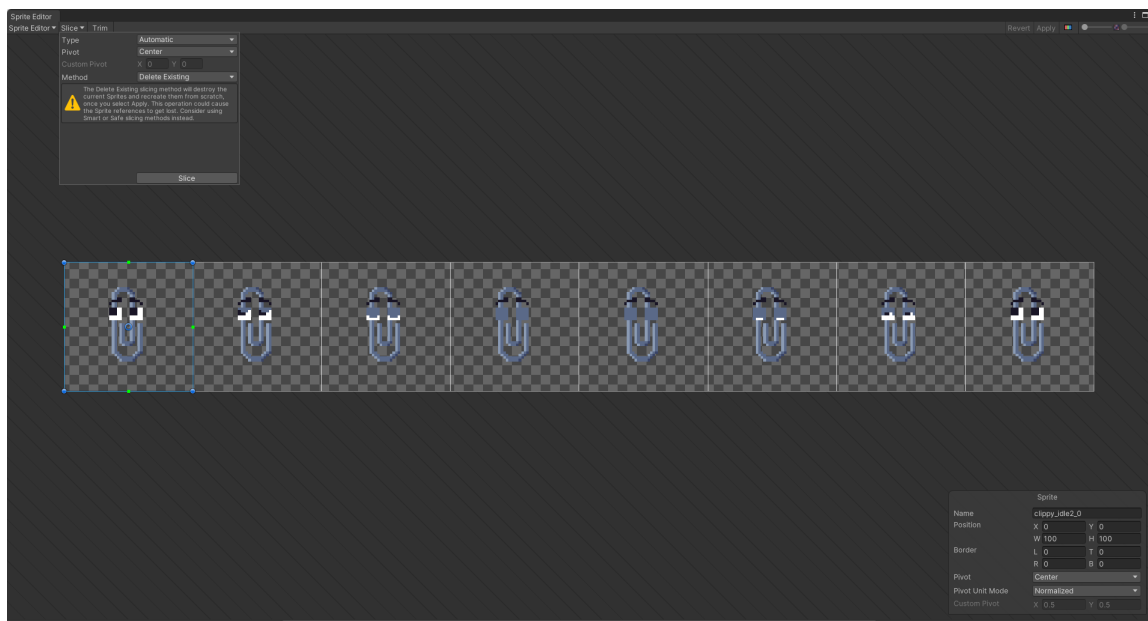
A karakter animációit különféle szkriptek felhasználásával oldottam meg ebben a játékban. A Sprite-okat mind egy mappába rendezzük, hogy később könnyen elérjük őket. Elsősorban előkészítjük a sprite-okat az animációhoz. Ehhez ki kell válasszuk a sprite sheet-et amelyet animálni szeretnénk, majd végrehajtunk rajta néhány beállítást. Legfontosabb lépés, hogy textúra típusnak a *Sprite (2D and UI)*, illetve sprite módnak a *Multiple* opciókat válasszuk.

Mivel pixelart stílusban dolgozunk, elengedhetetlen, hogy az egységenkénti pixelszámot 1-re állítsuk, a szűrési módot *Point(no filter)* opcióra, illetve a tömörítést *None* opcióra váltunk, hogy a megalkotott grafikánk tényleg úgy jelenjen meg, ahogy azt elképzeltük. (Ezeket a beállításokat a 4.3 ábrán is láthatjuk)



4.3. ábra. Clippy grafikai beállításai

Most, hogy a képek megjelenési beállításain túl vagyunk, itt az ideje, hogy felosszuk a sprite sheet-jeinket. A sprite editor gombra kattintva megnyílik a sprite szerkesztő felület, ahol pontosan be tudjuk állítani, hogy hol választódik el 2 kép egymástól a sprite sheet-ben, ezen felül pedig itt tudjuk a kép technikai széleit is beállítani, ami a tipprendszer szövegbuborék beállításainál volt nagyon hasznos.



4.4. ábra. Clippy második idle animációjának a felosztása

A sprite-ok előkészületét követően, kezdődhet az animációs szkriptek írása. Meghatározzunk néhány megkötést az animációval kapcsolatban, mint például az animációnk framerate-jét (filmkocka számát), véletlenszerűen történő animációk időkorlátját, vagy akár az animáció végét váró tiltó változók (Néhány példa a 4.1 kódrészletben). Mivel a Unity túl hamar frissít a retró stílusú játékunkhoz, a framerate-et az eredeti frissítési intervallum egy töredékére állítjuk.

```
public float idle2Interval = 30f; // Delay between each chance roll of idle2

public float frameRate = 0.1f; // Delay between each frame

private bool _allowedToClick = false; // Blocks clicking before animation ends

private bool _isHovering = false; // Bool variable for the hover animations
```

4.1. kódrészlet. Clippy különböző animációi állapotkezelőként megoldva

Ahhoz, hogy több animációt is hozzá tudjunk kötni a karakterünkhöz, illetve, hogy ezeket az animációkat szépen, rendszerezve tudjuk módosítani, ha szükséges, minden animációra egy külön függvényt írunk, majd felállítjuk az animációk logikáját.

```
private enum AnimationState {

    Idle1,

    Idle2,

    HoverIn,
```

```

        HoverOut
    }

    private AnimationState currentState = AnimationState.Idle1;

    private void Update() {

        switch (currentState) {

            case AnimationState.Idle1:

                PlayIdle1Animation();

                break;

            case AnimationState.Idle2:

                PlayIdle2Animation();

                break;

            case AnimationState.HoverIn:

                PlayHoverInAnimation();

                break;

            case AnimationState.HoverOut:

                PlayHoverOutAnimation();

                break;

        }

    }
}

```

4.2. kódrészlet. Clippy különböző animációi állapotkezelőként megoldva

Miután eldöntöttük, hogy karakterünknek milyen animációi lesznek, neki is foghatunk a szkriptek implementálásának. Minden animációnak létrehozunk egy változót, amely az időt fogja mérni. Ez a változó minden frissítéskor lekérdezi az aktuális időt, és hogyha ez az idő meghaladta az általunk beállított framerate-et, akkor új képkockát töltünk be, illetve visszaállítjuk az időváltozót, így elérve a kívánt képkocka frissítési időt.

```

private void PlayIdle1Animation() {

    _idle1Timer += Time.deltaTime;

    _image.sprite = idle1Frames[_idle1FrameIndex];

    _idle1Timer += Time.deltaTime;

    if (_idle1Timer >= frameRate) {

        _idle1FrameIndex = (_idle1FrameIndex + 1) % idle1Frames.Length;

        _idle1Timer = 0f;

    }

}

```

4.3. kódrészlet. Clippy alapértelmezett idle animációjának a szkript része

```

private void PlayHoverInAnimation() {

    _hoverTimer += Time.deltaTime;

    if (_hoverInFrameIndex < hoverInFrames.Length) {

        _image.sprite = hoverInFrames[_hoverInFrameIndex];

        _hoverTimer += Time.deltaTime;

        if (_hoverTimer >= frameRate) {

            _hoverInFrameIndex++;

            _hoverInLastFrameIndex = _hoverInFrameIndex;

            _hoverTimer = 0f;

        }

    }

}

```

```

else {

    if (_isHovering && _hoverInFrameIndex == hoverInFrames.Length) {

        _allowedToClick = true;

    }

}

}

private void PlayHoverOutAnimation() {

    _hoverTimer += Time.deltaTime;

    if (_hoverOutFrameIndex < hoverOutFrames.Length) {

        _image.sprite = hoverOutFrames[_hoverOutFrameIndex];

        _hoverTimer += Time.deltaTime;

        if (_hoverTimer >= frameRate) {

            _hoverOutFrameIndex++;

            _hoverTimer = 0f;

        }

    }

    else {

        currentState = AnimationState.Idle1;

        _idle1FrameIndex = 0;

        _hoverOutFrameIndex = 0;

    }

}

public void OnPointerEnter(PointerEventData eventData) {

    if (currentState != AnimationState.HoverIn) {

        currentState = AnimationState.HoverIn;
    }
}

```

```

        _hoverInFrameIndex = 0;

        _hoverOutFrameIndex = 0;

        _hoverTimer = 0f;

        _isHovering = true;

        _allowedToClick = false;
    }
}

public void OnPointerExit(PointerEventData eventData) {

    if (!_tips.activeSelf && currentState != AnimationState.HoverOut) {

        currentState = AnimationState.HoverOut;

        _hoverOutFrameIndex = hoverOutFrames.Length - _hoverInLastFrameIndex -
            1;

        _hoverOutFrameIndex = _hoverOutFrameIndex < 0 ? 0 :
            _hoverOutFrameIndex;

        _hoverInFrameIndex = 0;

        _hoverTimer = 0f;

        _isHovering = false;

        _allowedToClick = false;

    }
}

```

4.4. kódrészlet. Clippy kurzor alatti animációinak logikája

4.5. A puzzle elemek összeállítása

Ebben a fejezetben megtanuljuk, hogyan állítsuk össze a különféle puzzle-eket a játékunkban. Jelenleg, ami a képernyőn van, az olyan, mint egy festmény, nincsenek interakcióra kész objektumok, nincs semmi, amivel a játékos kezdhethetne valamit. Itt kerül sor a különféle funkciók, illetve mechanikák létrehozására, a puzzle-ek kiötlelésére és megvalósítására.

Figyelembe kell veyük a különböző puzzle játék stílusokat, mint például a kirakós játékok, a három találatos játékok, logikai rejtvények stb. Ki kell választanunk a nekünk legmegfelelőbbet, majd meg kell értenünk a puzzle játékunk lényegét. Ezzel a játékkal alapszintű programozást szeretnénk tanítani teljesen és/vagy részlegesen kezdő programozóknak, így az alapoktól kell kezdenünk, és letisztult, intuitív módon kell létrehozni a pályáinkat.

Meg kell határozzuk a puzzle-játék játékmenetét és szerkezetét. Gondoljunk a szintek előrehaladására, a nehézségi görbére, valamint az esetleges bónuszokra, amelyeket be szeretnénk építeni. Hozzunk létre egy sor alapvető mechanikát, amelyek a játék során következetesek maradnak, miközben új elemeket vezetünk be, hogy a játék továbbra is izgalmas maradjon. Gondoljuk át, hogy a rejtvényeket hogyan fogjuk bemutatni a játékosoknak, legyen az egy rácsalapú rendszer, egy narratíva-alapú megközelítés, vagy bármilyen más kreatív módszer.

A mi játékunk egy hiányos kódkiegészítő puzzle játék lesz. A képernyőn látható kódrészletből hiányozni fognak kulcsfontosságú elemek, mint például egy szám, egy változó deklaráció, vagy akár egy függvényfejléc.

```
#include <iostream>

int main() {

    std::cout << [ ] << std::endl;

    return 0;

}
```

4.5. ábra. "Hello World!" pálya

Meghatároztuk már a játékunk stílusát, a puzzle-k megoldási folyamatát, itt az ideje megtervezni az elemeket, amivel interaktálni fog a játékos. A játékos többféle blokkelem közül választhat, miután feloldotta őket. Minden blokkelemnek meg lesz a saját szerepe, így kulcsfontosságú lesz, hogy a játékos elsajátítsa minden blokkelem tulajdonságát, hogy haladhasson a játékban.

```
[ ] % Value Test
int variable = [ ] ;
```

4.6. ábra. Néhány blokkelem a játékból

A puzzle játékok készítése során kulcsfontosságú szerepet tölt be a rejtvények megtervezése, amelyek igazodnak az előzőleg meghatározott mechanikákhoz és célokhoz. Törekedünk a kihívást jelentő, de mégis megoldható és tanító jellegű rejtvények közötti egyensúlyra. Biztosítjuk, hogy minden rejtvénynek egyértelmű megoldása legyen, illetve mindenféle kétértelműség vagy tisztességtelenség nélkül megoldható legyen.

A játék előrehaladtával fokozatosan vezetünk be új rejtvényelemeket, mechanikákat, illetve akadályokat, hogy fenntartsuk a játékos érdeklődését. Mivel sokféle blokkelem elérhető a játékban, más és más szereppel, nagyon fontos, hogy ne vezessünk be egyszerre túl sok új elemet, viszont ne használjuk ugyanazokat az elemeket huzamosabb ideig. Meg kell találnunk az

egyensúlyt, hogy a játékos ne érezze túlterhelve magát, hogy tanulhasson és szórakozhasson a játékunkba, viszont azt sem szeretnénk, hogy a játék monoton és unalmas legyen.

```
#include <iostream>

int main() {

    int num = 10;

    if (num > 0) {
        std::cout << "The number is positive." << std::endl;
    } else if (num < 0) {
        std::cout << "The number is negative." << std::endl;
    } else {
        std::cout << "The number is zero." << std::endl;
    }

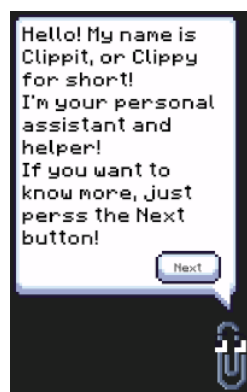
    return 0;
}
```

4.7. ábra. "if-elseif-else" pálya

4.6. A tipprendszer olyan, mint egy rejtett oktató

Egy tipprendszer beépítése egy puzzle játékba számos előnnyel járhat. A rejtvenyjátékok nehézsége változó lehet, egyes játékosok bizonyos rejtvényeket kihívásnak találhatnak, vagy bizonyos pontokon elakadhatnak, míg mások könnyedén átszálnak a feladat fölött, így nagyon fontos, hogy a tipprendszerünk segítőkész legyen, de ne irritálja a játékosokat.

A tipprendszer biztonsági hálóként működhet, segítséget nyújtva a nehézségekkel küzdő játékosoknak. A segítségek felajánlásával a játék szélesebb közönség számára válik elérhetővé, beleértve az alkalmi játékosokat, vagy azokat, akiknek nincs tapasztalatuk a kódolásban, egyúttal pedig ösztönözhetik is a játékost a tovább haladásra.

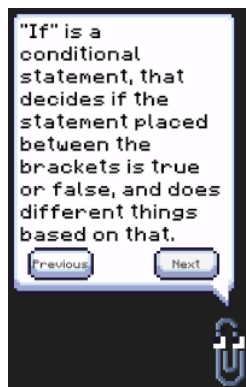


4.8. ábra. Clippy, a játékos személyi asszisztense

A rejtvenyjátékok gyakran támaszkodnak a teljesítmény és a fejlődés érzésére, ahogy a játékosok megoldják a rejtvényeket és haladnak előre a szinteken, viszont, ha a játékosok elakadnak

egy különösen nagy kihívást jelentő rejtvényen, egy tipprendszer egy lökést adhat nekik a helyes irányba, segítve őket az akadály leküzdésében és a továbblépésben.

A tippek biztosításával a játékosok megtapasztalhatják a teljesítmény érzését anélkül, hogy túlterheltnék vagy túl sokáig elakadtnak éreznék magukat, ez növeli a játékosok elkötelezettségét és motivációját, mivel tovább haladhatnak a játékban, és újabb kihívásokat fedezhetnek fel.



4.9. ábra. Információs tipp

Kulcsfontosságú ugyanakkor, hogy egyensúlyt teremtsünk a kihívás és a frusztráció között. Míg a kihívást jelentő rejtvények megoldása kifizetődő lehet, a túlzott nehézség a játékosok frusztrációjához és a játék elhagyásához vezethet.

Egy jól elkészített tipprendszer módot kínál a frusztráció enyhítésére azáltal, hogy tippeket vagy megoldásokat ad a játékosoknak, amikor valóban elakadnak.

A fokozatos tipprendszer beépítésével, amely finom utalásokkal kezdődik (4.10 ábra), majd egyre egyértelműbb útmutatáshoz vezet, a játékosok kiválaszthatják a kívánt segítség szintjét anélkül, hogy teljesen elárulnák a megoldást. Ez a megközelítés segít fenntartani a kihívást jelentő élményt, miközben a játékosok elkötelezettek és motiváltak maradnak a rejtvények megoldására.



4.10. ábra. Segítőkéz tipp

A rejtvényjátékok gyakran egyedi mechanikát tartalmaznak, és megkövetelik a játékosoktól a kritikus gondolkodást, a minták elemzését és a problémamegoldó készségek fejlesztését. tipprendszer értékes tanulási eszközként szolgálhat, megtanítva a játékosoknak a rejtvények megoldásának különböző stratégiáit és megközelítéseit.

A tippek betekintést nyújthatnak a rejtvény mögöttes logikájába vagy szabályaiba, így a játékosok tanulhatnak belőlük, és a jövőben hasonló kihívásoknál alkalmazhatják a tudást.

Ahogy a játékosok egyre gyakorlottabbá válnak, egyre kevésbé támaszkodhatnak a tipp-rendszerre, így a játékosok a mesteri tudás érzését és elégedettséget érezhetik a fejlettebb képességeikben.



4.11. ábra. Dicsérő tipp

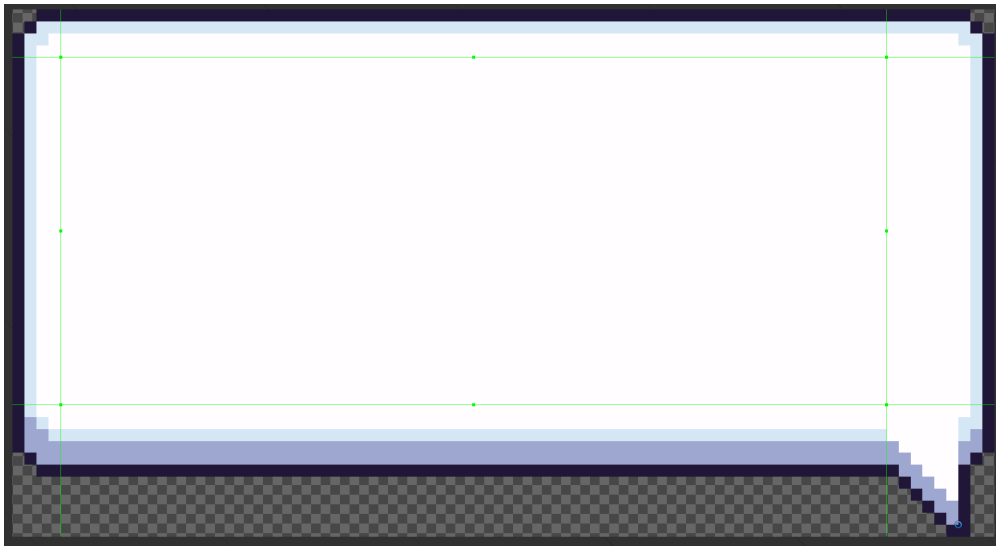
Egy tipprendszer egy puzzle játékban elengedhetetlen a hozzáférhetőség biztosításához, a kihívás és a frusztráció egyensúlyának megteremtéséhez, a játékosok fejlődésének elősegítéséhez, a készségfejlesztés elősegítéséhez, és végső soron a játékosok elkötelezettségének, megtartásának és elégedettségének biztosításához. Szükség esetén segítséget nyújtva javíthatja az általános élményt, miközben fenntartja a kirakós játékok jutalmazó és intellektuálisan ösztönző jellegét.

4.7. A tipprendszer megalkotása

Mivel előzőleg láhattuk, a játékba implementált tipp rendszer szövegbuborék formájában jelenik meg Clippy fölött. Ez a dizájn ötlet rengeteg komplikációt vezetett fel. Végig kellett gondolni, hogyan is fog kinézni az a szövegbuborék, konstans lesz-e a mérete, vagy hogyan fog a játékos navigálni a tippek között.

Ezek mind nagyon fontos döntőpontok voltak a tervezés közben. Végezetül úgy döntöttem, hogy a szövegbuboréknak méreteződnie kell a szöveg hosszúságához, illetve, hogy a játékos gombokkal fog váltani a tippek között.

Ahogy előzőleg is említettem, a sprite editor segítségével sikerült megoldani a szövegbuborék méretezését. A sprite editor segítségével be lehet állítani a képek technikai szélét, amit újra méretezéskor hagyjon ki, ezáltal a szövegbuborék széle nem fog deformálódni. Ugyanitt át kell állítani a pivot pontot a szövegbuborék szárához, hogy a programunk tudja merre tudja növelni a szövegbuborék méretét.



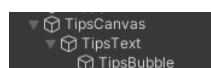
4.12. ábra. A szövegbuborék beállítása a Sprite Editor ablakban

Ahhoz, hogy a szövegbuborék automatikusan újra méreteződjön a szöveg mennyisége alapján, szükség lesz még egy pár beállításra.

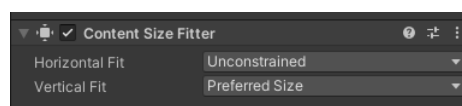
Elsősorban fel kell állítanunk a tipp szöveg objektum és a tipp buborék objektum között egy szülő-gyerek kapcsolatot (lásd [4.13](#)).

Másodsorban, be kell állítanunk a tipp szöveg objektumra a dinamikus méretet. Mivel nem szeretnénk, hogy vízszintesen újraméreteződjön a szövegdoboz, így arra a paraméterre statikus értéket adunk meg, míg függőlegesen *Preferred Size* opcióra állítjuk, hogy újra méretezze magát a szöveg mennyisége alapján ([4.14](#) ábra).

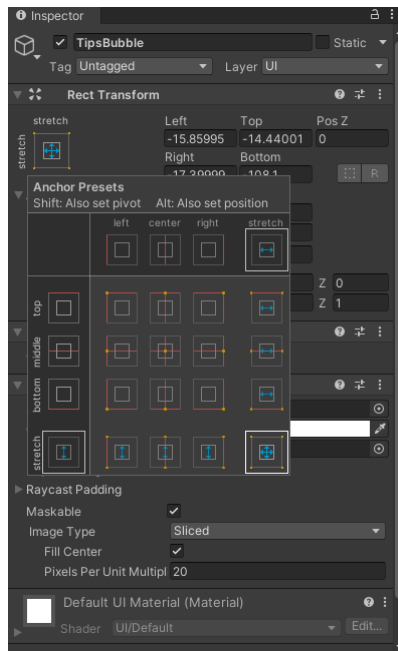
Végül, de nem utolsó sorban, a horgonyok beállítási menüben a Shift + Alt billentyűkombináció lenyomásával, illetve a jobb alsó sarokban lévő opciót választva a szövegbuborék automatikusan a szülő objektumhoz igazítódik. Ezután már csak be kell állítani mind a 4 oldalra a padding-et, amivel megakadályozzuk, hogy a szöveg kifolyjon a szövegbuborékból ([4.15](#) ábra).



4.13. ábra. Szülő - gyerek kapcsolat a tipp objektumok között



4.14. ábra. Szöveghez igazodó objektum kikötések



4.15. ábra. Szövegbuborék méretezése a szöveg mennyisége után

A grafikai megjelenítés letudása után jöhetett a szkriptek megírása. Mivel a tippszövegek a vizuális felületen (Inspector ablak) vannak hozzáadva a pályákhoz, így a tippeket kezelő szkriptek elég egyszerűek, viszont akármennyire is egyszerűek, mégis kulcsfontosságúak a tipprendszerben.

```
void Update()
{
    previousButton.SetActive(_tipIndex != 0);
    nextButton.SetActive(_tipIndex < tipLines.Count - 1);
    _text.SetText(tipLines[_tipIndex]);
    textBubble.SetText(tipLines[_tipIndex]);
}

private void ShowPreviousTip()
{
    if (_tipIndex > 0) {
        --_tipIndex;
    }
}
```

```
    }  
}  
  
private void ShowNextTip()  
{  
    if (_tipIndex < tipLines.Count - 1) {  
        ++_tipIndex;  
    }  
}
```

4.5. kódrészlet. Típrendszer kezelőszkriptjei

Összefoglaló

Dolgozatomban egy 2D puzzle játék megtervezésével és implementálásával foglalkoztam, amelyet a Unity keretrendszeren belül valósítottam meg. Először ismertettem a videójátékok rövid történelmét, fontosságát, később beszéltem a 2D és a 3D játékok közötti különbségről, bemutattam a felhasznált technológiákat, majd részletes leírást adtam egy 2D puzzle játék elkészítéséről. Kellő részletességgel átvettük a Unity használatát, hogy a dolgozatom elolvasása után könnyebb legyen nekivágni a játékfejlesztésnek. A továbbiakban ismertettem a játékunk célját, tervezési folyamatát, illetve magát az implementációt is. Ezek után részletesen is megnéztük, hogy milyen dizájn nyelvet, technikákat és praktikákat használtam a grafikai elemek megtervezésében, létrehozásában és animálásában. A játék karakterének, Clippynek a létrehozását több kategóriára is felosztottam. Az első kategóriában ismertettem a gondolatmenetet, mégis miért pont Clippy lett a választott karakter. Emellett megnéztük, hogy milyen módszerekkel lehet egy játék béli karaktert életre kelteni a Unity által kínált technológiák felhasználásával. Volt szó a sprite sheet-ek megtervezéséről és beállításáról, hogy könnyedén felhasználhassuk őket az animációkban. Mindemellett említésre kerültek az általam írt szkriptek is, amelyek életet adtak ezeknek az animációknak. Végül, de nem utolsó sorban kitértem a tipprendszer megvalósítására, is, milyen gondolatmenet van mögötte, mi ösztönzött arra, hogy tipprendszert implementáljak a projektbe, hogyan lett megtervezve, mi több létrehozva a tippek vizuális felülete, illetve mi alapján írtam meg magukat a tippeket.

Jövőbeli terveimet illetően szeretnék jobban elmerülni a videójátékok tervezésében és fejlesztésében. Szeretnék több figyelmet fordítani a technológiai részére, mindemellett szeretném fejleszteni a dizájnolási képességeimet is, ugyanakkor érdekelt vagyok az animáció világában is. Továbbá érdemesnek tartanám a játékok integrálását az oktatási közegbe. Véleményem szerint, megvan az a réteg akiknek egyszerűbb, szórakoztatóbb, ráadásul akár hatékonyabb is lenne a tanulás hasonló videójátékok segítségével.

GitHub— <https://github.com/AlgorythmicsUniverse/game>

Ábrák jegyzéke

2.1. 2D illetve 3D karakter hasonlítás	12
3.1. Unity felhasználói felület	14
3.2. Hierarchia ablak	14
3.3. Projekt ablak	15
3.4. Konzol ablak	15
3.5. Inspector ablak	16
3.6. Scene ablak	16
3.7. Játék ablak	17
3.8. Objektum létrehozása	18
3.9. Kép komponens	18
3.10. Komponens hozzáadása	18
3.11. Projekt rendezettsége	19
3.12. Eszközsáv	19
4.1. Visual Studio Code pixelart változata	23
4.2. Clippy második idle animációjának a sprite sheet-je	25
4.3. Clippy grafikai beállításai	25
4.4. Clippy második idle animációjának a felosztása	26
4.5. "Hello World!" pálya	31
4.6. Néhány blokkelem a játékból	31
4.7. "if-elseif-else" pálya	32
4.8. Clippy, a játékos személyi asszisztense	32
4.9. Információs tipp	33
4.10. Segítőkész tipp	33
4.11. Dicsérő tipp	34
4.12. A szövegbuborék beállítása a Sprite Editor ablakban	35
4.13. Szülő - gyerek kapcsolat a tipp objektumok között	35
4.14. Szöveghez igazodó objektum kikötések	35
4.15. Szövegbuborék méretezése a szöveg mennyisége után	36

Irodalomjegyzék

- [BSPW19] Nancy Baym, Limor Shifman, Christopher Persaud, and Kelly Wagman. Intelligent failures: Clippy memes and the limits of digital assistants. *AoIR Selected Papers of Internet Research*, 2019.
- [GLBC20] SAMUEL García-Lanzo, Ivan Bonilla, and Andres Chamarro. The psychological aspects of electronic sports: Tips for sports psychologists. *International Journal of Sport Psychology*, 51(6):613–625, 2020.
- [Haa14] John K Haas. A history of the unity game engine. *Diss. Worcester Polytechnic Institute*, 483(2014):484, 2014.
- [May19] Richard E Mayer. Computer games in education. *Annual review of psychology*, 70:531–549, 2019.
- [Tur95] Alan M Turing. Lecture to the london mathematical society on 20 february 1947. *MD COMPUTING*, 12:390–390, 1995.