JavaScript u kontekstu domene edukativnih igara

Dominiković, Luka

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
Ak. god. 2021./ 2022.

Luka Dominiković

JavaScript u kontekstu domene edukativnih igara

Diplomski rad

Mentor: dr.sc. Kristina Kocijan, izv. prof.

Zagreb 2022.

Izjava o akademskoj čestitosti

Izjavljujem i svojim potpisom potvrđujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na istraživanjima te objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

_____

(potpis)

Sadržaj

Izjava o akademskoj čestitosti 1
Sadržaj 2
1. Uvod 3
2. Igre u kontekstu edukacije 5
2.1. Kviz znanja 8
2.2. Igra memorije 8
2.3. Klizna slagalica 9
2.4. CARNETovi prijedlozi igara 10
3. JavaScript u kontekstu izrade igara 14
4. Kreacija kviza znanja 16
5. Kreacija igre memorije 28
6. Kreacija klizne slagalice 43
7. Zaključak 71
8. Literatura 73
Sažetak 76
Abstract 77

1. Uvod

U ovom radu će se pisati o jednom aspektu edukacije koji se često izbjegava u nastavi, a to je korištenje edukativnih igara. Edukativne igre su veoma koristan alat koji omogućuje učenje kroz igru te može pomoći učenicima da im iskustvo u nastavi bude zabavnije od monotonog načina na kakav su navikli. No ovaj rad se neće fokusirati na važnost edukativnih igara u nastavnom procesu, već ovim radom želimo istražiti može li se jedan od najpopularnijih programskih jezika[1] koristiti u svrhu izrade upravo edukativnih igara. Srećom CARNET daje prijedloge edukativnih igara koje su poželjne u nastavi tako da se možemo odlučiti na one najkorištenije ili najzabavnije od ponuđenih.

Za početak ćemo spomenuti igre u kontekstu edukacije, upoznat ćemo se s određenim vrstama igara (kviz znanja, igra memorije i klizna slagalica) te vidjeti kakve zaključke donose istraživanja koja su se bavila tim tipovima igara. U istom poglavlju će se prikazati i alati koje CARNET preporučuje za izradu edukativnih igara.

Zatim ćemo se dotaknuti teme programskih jezika. Vidjet ćemo koji se jezici većinom koriste u izradi igara te ćemo se bliže upoznati s JavaScript programskim jezikom kojeg ćemo koristiti u svrhu ovog istraživanja. U ovom poglavlju ćemo naći i hipotezu samog rada, a to je da smatramo da se JavaScript može koristiti za izradu edukativnih igara odabranih za ovo istraživanje.

U sljedećem poglavlju ćemo pokušati kreirati najkorišteniju vrstu edukativne igre, a to je kviz znanja. Iako mnogi nastavnici ne koriste edukativne igre u nastavi, ukoliko jesu, to je vrlo vjerojatno neka vrsta kviza, a budući da se JavaScript može koristiti za izradu anketa, ova vrsta igre ne bi trebala predstavljati veliki problem.

Nakon pokušaja kreiranja kviza znanja, prebacit ćemo fokus na izradu igre memorije. To je veoma korisna igra koja pomaže kod razvijanja sposobnosti koja je itekako

[1] Podatak s mrežnog izvora - https://www.devjobsscanner.com/blog/top-8-most-demanded-languages-in-2022/

potrebna učenicima, a to je pamćenje. Ova vrsta igre je puno zahtjevnija za izradu, ali je i zabavnija za korištenje te omogućuje praćenje učenikovih sposobnosti preko broja poteza koji im je potreban za rješavanje određenog problema.

U šestom poglavlju ćemo pokušati kreirati kliznu slagalicu, vrstu edukativne igre koja može pomoći učenicima kod predmeta koji svoje znanje trebaju pridružiti nekom vizualnom elementu. Zadivljujuće kompleksna igra koja je također i najteža za izraditi od tri igre koje ćemo pokušati kreirati u ovom radu. Na posljetku ćemo se osvrnuti na postavljenu hipotezu u kontekstu istraživanja i dati kratak pregled rada.

2. Igre u kontekstu edukacije

Mrežne igre se zahvaljujući razvoju HTML5 jezika, u posljednjih desetak godina, razvijaju iznimno brzo, a razlog tomu je opisan u Hawkesovoj knjizi iz 2011. godine u kojoj objašnjava nove funkcije uvedene u HTML5 standard. Hawkes (2011) tvrdi da nove funkcije omogućavaju lakši način integracije JavaScript jezika u HTML, te samim time lakšu izradu mrežnih igara. Edukativne igre se, iako nisu novost, sve više koriste u nastavi. Razlog tomu je razvoj same tehnologije koja se sve više koristi u obrazovanju. Gordon već 1970. godine govori o korištenju igara za razvoj unutar učionice te smatra da bi bile iznimno korisne za motivaciju učenika (Gordon, 1970).

Problem koji se danas pojavljuje je što su edukativne igre digitalizirane, a proizvođači igara financiraju samo projekte koji će zaraditi što više novca te zanemaruju edukativnu stranu i fokusiraju se na zabavu. Autori Mester, Molcer i Delic 2011. godine objavljuju rad u kojem ističu upravo taj problem te daju svoje prijedloge za izradu edukacijskih igara te kako one mogu biti i financijski isplative. Naime, edukativne igre su jednostavnije i jeftinije za izraditi od onih koje se izrađuju isključivo za zabavu. Čak i igre koje nisu izrađenje s namjerom za edukaciju uspijevaju pružiti nova znanja.

O utjecaju edukativnih igara govori Annetta u svom radu iz 2008. godine gdje tvrdi da se igre mogu koristiti za poboljšanje edukativnog procesa. Same prednosti edukacijskih igara su objašnjene u radu iz 2020. godine gdje Zeng, Sparks i Shang dolaze do zaključka da edukacijske igre pomažu kod poticanja motivacije za učenje, poboljšavaju rezultate učenja, stvaraju okruženje za učenje te promiču transformaciju metoda učenja.

Iako se često spominju pozitivne strane korištenja edukativnih igara, potrebni su nam i rezultati korištenja igara u nastavi kako bi vidjeli jesu li one zapravo korisne ili ne. Prvo istraživanje koje ćemo spomenuti se provelo u Zagrebu 2006. godine u kojem se ispitivalo znanje učenika o uporabi slova č, ć, dž i đ. Ovo je prvo istraživanje koje ću opisati budući da se ne bavi direktno igrama, već samoj uporabi tehnologije u nastavi.

Pitanje na koje je ovo istraživanje pokušalo naći odgovor je može li se učenje ovih slova u školi poboljšati korištenjem tehnologije. Istraživanje su proveli Družijanić-Hajdarević, Vučković i Dovedan u osnovnoj školi u Zagrebu s učenicima šestog i osmog razreda. Za početak su proveli uvodno testiranje o njihovom znanju o uporabi već spomenutih slova. Nakon uvodnog testiranja, učenici su podijeljeni u dvije skupine, jedna je vježbala uz pomoć računala, a druga na klasičan način. Ono što je zaključeno ovim istraživanjem je da su učenici koji su vježbali uz pomoć računala postigli bolje rezultate od onih koji su vježbali na klasičan način (Družijanić-Hajdarević *et al.*, 2006). Ovo istraživanje je dosta dobar pokazatelj da se uz pomoć tehnologije može poboljšati nastavni proces.

Mubaslat (2012) opisuje istraživanje koje je provedeno da se vidi pomažu li video igre kod učenja stranog jezika. U tom istraživanju se dolazi do zaključka da video igre pomažu kod učenja stranog jezika, u ovom slučaju engleskog, posebno zato što stvaraju okolinu bogatu interakcijom među učenicima.

Drugo zanimljivo istraživanje na temu korištenja video igara u nastavi je pokušalo dokazati da se video igre mogu koristiti u svrhu učenja povijesti. U ovom istraživanju su autori Zirawaga, Olusanya i Maduku (2017) pokušali dokazati da se video igre, budući da se u njima moraju pratiti određena pravila, mogu koristiti kao pomoćni alat u samom nastavnom procesu. Oni smatraju da video igre mogu biti iznimno korisne u nastavi iz razloga što se mogu koristiti uz tradicionalne metode kako bi poboljšali iskustvo učenja te im mogu pomoći u razvoju raznih sposobnosti poput kritičkog mišljenja, kreativnosti, poštivanja pravila, adaptacije i mnogih drugih. Oni smatraju da učenje ne treba biti monotono i ne smije se svoditi samo na memoriranje sadržaja kojeg nastavnici tumače. Tijekom istraživanja su koristili razne vrste igara kao što su osmosmjerka, križaljka i klizna slagalica. Ovim istraživanjem su došli do zaključka da postoje brojne prilike u nastavnom procesu gdje se igre mogu koristiti kako bi poboljšali iskustvo učenika.

Još jedno istraživanje o korisnosti edukativinih igara koje ću spomenuti je ono koje su proveli Najdi i Sheikh (2012). U tom istraživanju su pokušali doći do zaključka čine li igre uopće razliku u edukaciji. Upravo to istraživanje je veoma korisno za potrebe ovoga rada

budući da su koristili vrste igara koje ćemo kreirati u ovom radu. Vrste igara koje su koristili su igre s karticama, poput igre memorije, digitalne vrste kvizova te razne vrste zagonetki i slagalica.

Korištenje tehnologije se također može primijeniti u području matematike (Al-Mashaqbeh i Al Dweri, 2014). Istraživanje je kao i većina ovakvih istraživanja provedeno tako da su učenici podijeljeni u dvije grupe te je njihovo znanje provjereno prije početka samog istraživanja. U ispitu predznanja su obje grupe postigle podjednake rezultate, ali nakon tog ispita jedna grupa se pripremala uz korištenje igara dok se druga grupa pripremala na tradicionalan način. U drugom ispitu je grupa koja se pripremala uz pomoć igara postigla bolje rezultate te su dokazali da je korištenje igara u nastavnom procesu dovelo do poboljšanja matematičkih vještina učenika.

Također je dobro napomenuti da emocije igraju veliku ulogu u samom procesu učenja. Istraživanje iz 2020. godine se temelji upravo na ulozi emocija u nastavnom procesu (Cheng, Huang i Hsu, 2020). To istraživanje koristi mrežnu igru kako bi utjecali na emocije učenika te vidjeli može li ta igra olakšati učenicima učenje znanstvenih koncepata. Cheng i suradnici (2020) u ovom istraživanju dolaze do jako zanimljivih zaključaka. Naime, ukoliko učenik osjeti samo pozitivne emocije, postiže bolje rezultate odmah nakon što je izložen igri, ali njegovi rezultati opadaju što vrijeme više prolazi. Oni učenici koji su osjetili i pozitivne i negativne emocije također postižu bolje rezultate nakon što su izloženi igri, ali njihovi rezultati ostaju bolji u duljem periodu što bi značilo da negativne emocije imaju dugoročnije posljedice, bile one pozitivne ili negativne.

Navedena istraživanja donose dokaze da je korištenje igara u edukaciji veoma dobar alat za promoviranje učenja novih vještina koje školski kurikulum zahtijeva. Uz pomoć brojnih istraživanja i prijedloga koje igre koristiti u edukaciji možemo zaključiti da su one zaista korisne kao alat unutar nastave, ali za potrebe ovog istraživanja ćemo se fokusirati na tri vrste igara, a to su kviz znanja, igra memorije te klizna slagalica o kojima će biti više riječi u nastavku ovog rada.

2.1. Kviz znanja

Znanje iz pojedinih predmeta u nastavi se provjerava uz pomoć raznih testova znanja koji se ocjenjuju te tako evaluiramo i pratimo znanje i razvoj učenika. Korištenjem kviza znanja možemo pomoći učenicima da ostvare bolje rezultate u nastavi ukoliko ih koristimo u svrhu ponavljanja gradiva prije samog ispita. U prilog tome govore i Aljezawi i Albashtawy koji su 2015. godine objavili rezultate istraživanja učinka kviza znanja na uspjeh učenika. To su istražili tako što su podijelili učenike u dvije grupe, od kojih je jedna za ponavljanje gradiva koristila igru u obliku kviza, dok je druga ponavljala gradivo u obliku lekcije. Autori su napomenuli da su obe grupe imale jednak uspjeh prije njihovog istraživanja, ali rezultati samog istraživanja su pokazali da su po završetku eksperimenta učenici iz grupe koji su koristili kviz bili zadovoljniji tom metodom i da su postizali bolje rezultate na ispitu koji je usljedio.

Još jedno istraživanje koje pruža dobar argument za kroištenje kviza znanja je istraživanje iz 2008. godine u kojem Wang dolazi do zaključka za se digitalni kviz znanja može koristiti u nastavi kao način provjere znanja umjesto standardnog pismenog ispita. Po završetku istraživanja dolazi do zaključka da se učenicima sviđa takva vrsta provjere znanja te da se uz korištenje adekvatne tehnologije takva vrsta provjere znanja zaista može iskoristit u određenim situacijama unutar nastavnog procesa. Da se ne radi samo o izoliranim eksperimentima mogli smo vidjeti u ne tako davnoj nastavnoj situaciji uzrokovanoj COVID virusom, kada je ovakva vrsta provjere bila korištena u obrazovanju širom svijeta.

2.2. Igra memorije

Igra memorije je jednostavna igra u kojoj se kartice nalaze okrenute licem prema dolje te je potrebno okrenuti dvije jednake kartice koje zatim ostanu okrenute do kraja igre, a igra završava kada su sve kartice okrenute.

Ovaj tip igre poboljšava rad mozga i pozitivno utječe na njegovo zdravlje, a istraživanje kojim se to može dokazati je izvedeno 2008. na sveučilištu u Michiganu, gdje su Heeter, B. Winn, J. Winn i Bozoki željeli saznati pomaže li igra memorije kod zdravlja mozga. U tom istraživanju su došli do zaključka da je rad mozga poboljšan većini sudionika. Doduše, to istraživanje nije provedeno nad učenicima fakulteta već na starijim osobana između 60 i 80 godina.

Istraživanje provedeno nad učenicima osnovnih škola proveo je doktor Sivakumar (2022) koji je htio istražiti pomaže li igra memorije u razvoju učenika. U svom istraživanju koristio je razne vrste igara koje utječu na mozak, ali igra memorije je pokazala najbolje rezultate te je zaključio da učenici koji su koristili igru memorije postižu bolje rezultate na ispitima nakon eksperimenta od onih koji nisu.

2.3. Klizna slagalica

Razne vrste slagalica se mogu koristiti u edukaciji iz razloga što pomažu kod napretka vještina rješavanja problema, pobošljavaju percepciju te pobošljavaju procjenu strategija.

Postoje istraživanja koja dokazuju tu tvrdnju, a jedno od njih se odvilo na sveučilištu u Malagi još 2003. godine u kojem su Aguilera i Mendiz htjeli dokazati da igre nisu samo gubitak vremena te da se mogu koristiti u nastavi kako bi pobošljali iskustvo učenika i pomogli im kod razvijanja određenih vještina. Prema njihovom istraživanju najbolje rezultate pružaju razne vrste slagalica u oblicima video igara.

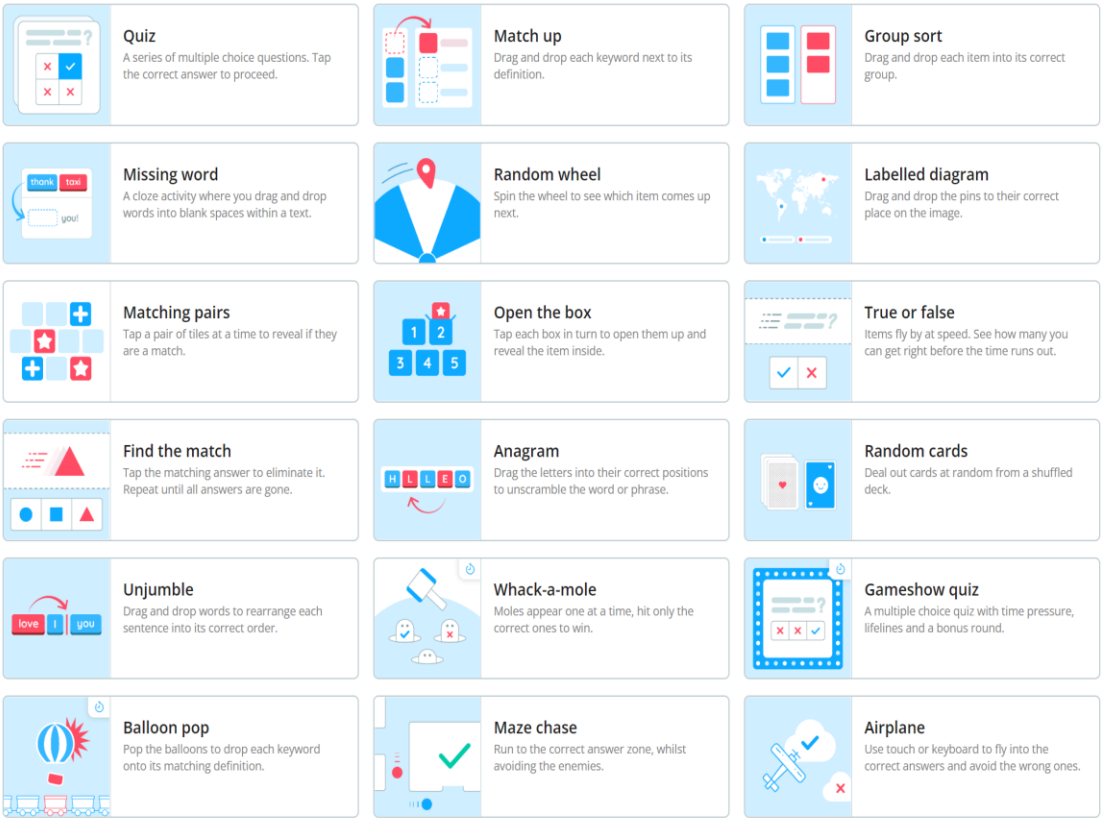
Drugo istraživanje o upravo ovoj vrsti igara se odvilo na sveučilištu u Sofiji 2019. godine gdje su Antonova i Bontchev došli do zaključka da se korištenjem slagalica u nastavi može poboljšati razvoj kompleksnih vještina kod učenika.

Budući da postoji ogroman broj raznih slagalica, za potrebe ovog rada sam se odlučio za kliznu slagalicu, vrstu igre u kojoj se na ploči sastavljenoj od 9 polja treba složiti slika. Jedno polje na ploči je uvijek prazno kako bi se omogućilo pomicanje ostalih dijelova.

Odlučio sam se za ovu vrstu igre iz razloga što smatram da je dovoljno kompilicirana da se na njenoj izradi mogu istražiti ograničenja programskog jezika kojim ćemo se služiti.

2.4. CARNETovi prijedlozi igara

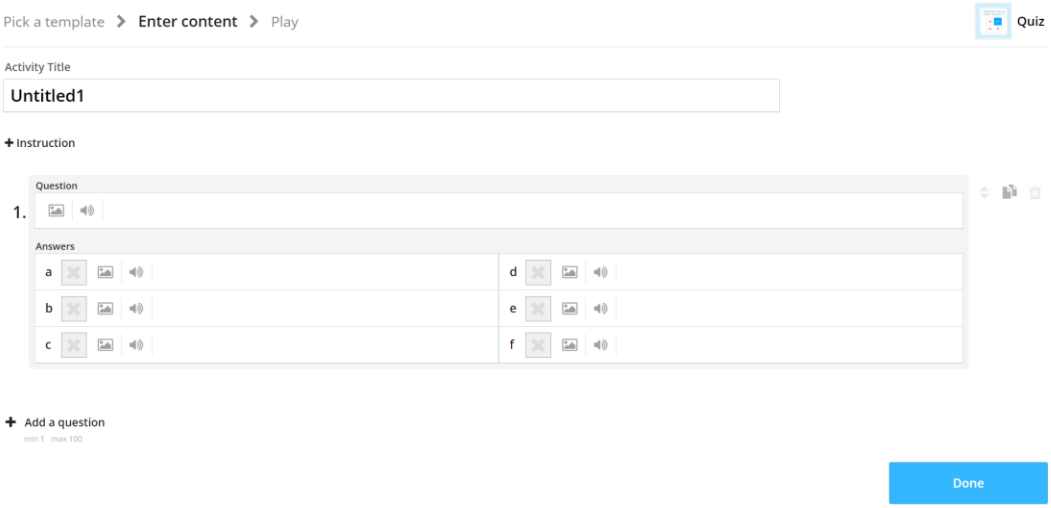
CARNET na svojoj mrežnoj stranici[2] daje popis igara koje predlažu za korištenje u nastavi. Od njihovih prijedloga ćemo se fokusirati na Wordwall[3], Match the Memory[4] te Puzzel.org[5] alate za izradu igara. Tim alatima CARNET pokušava olakšati izradu određenih tipova igara koje nastavnici mogu koristiti u nastavnom procesu.



Slika 1 – Wordwall odabir vrste igre

[2] https://e-laboratorij.carnet.hr/category/igre/
[3] https://wordwall.net
[4] https://matchthememory.com
[5] https://puzzel.org

Kao što je prikazano na slici 1, Wordwall alat omogućuje kreaciju raznih interaktivnih igara. Jedna od njih je upravo kviz znanja (slika 2) koji je i u fokusu ovog rada.

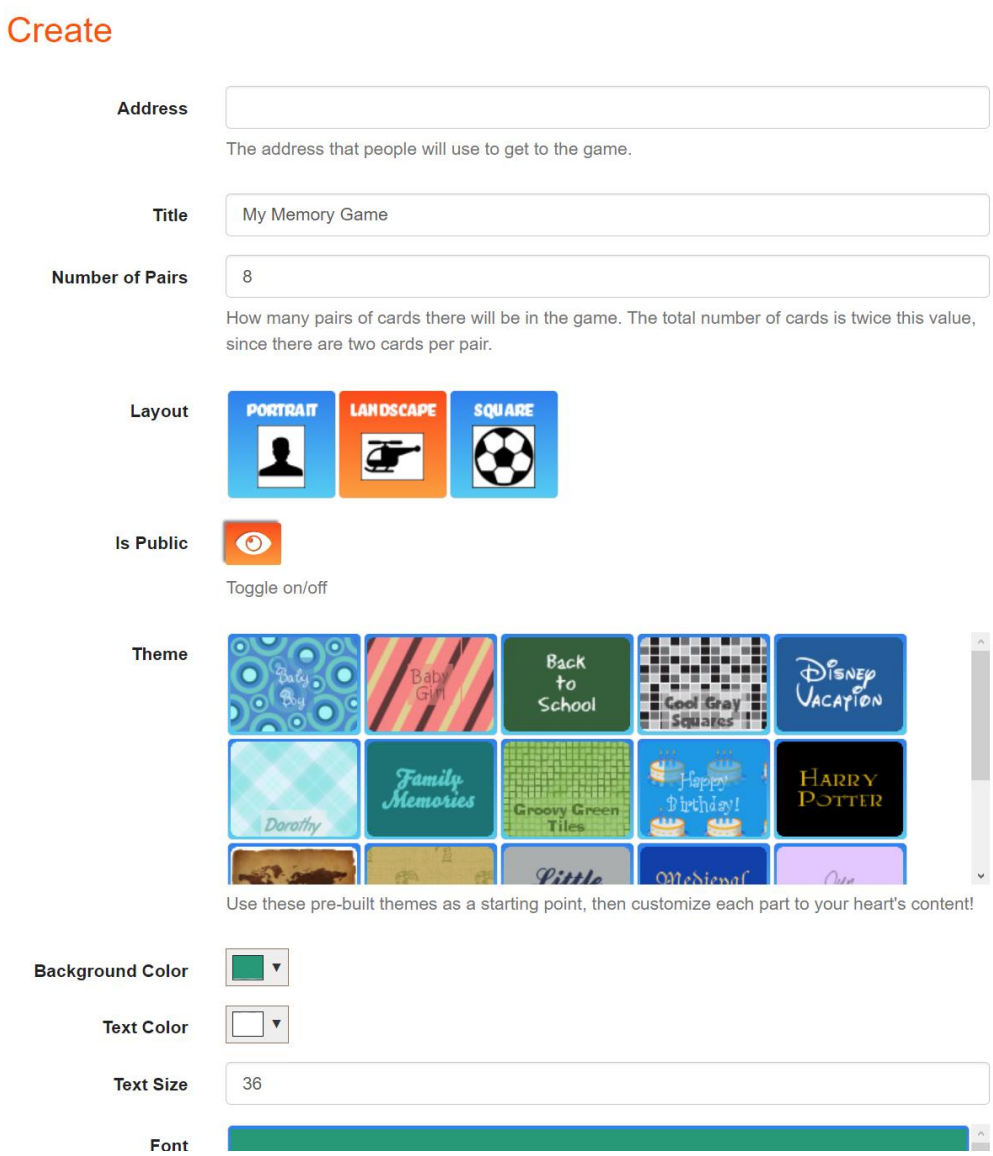


Slika 2 – Wordwall izrada kviza znanja

Slika 2 prikazuje sučelje za izradu kviza znanja unutar samog alata. Možemo vidjeti da nam alat omogućava odabir naslova samog kviza, postavljanje pitanja kojem možemo dodati sliku ili zvuk, dodavanje višestrukog izbora odgovora te dodavanje novih pitanja. Na *About*[6] stranici samog alata možemo vidjeti da se igre izrađuju uz pomoć C# programskog jezika.

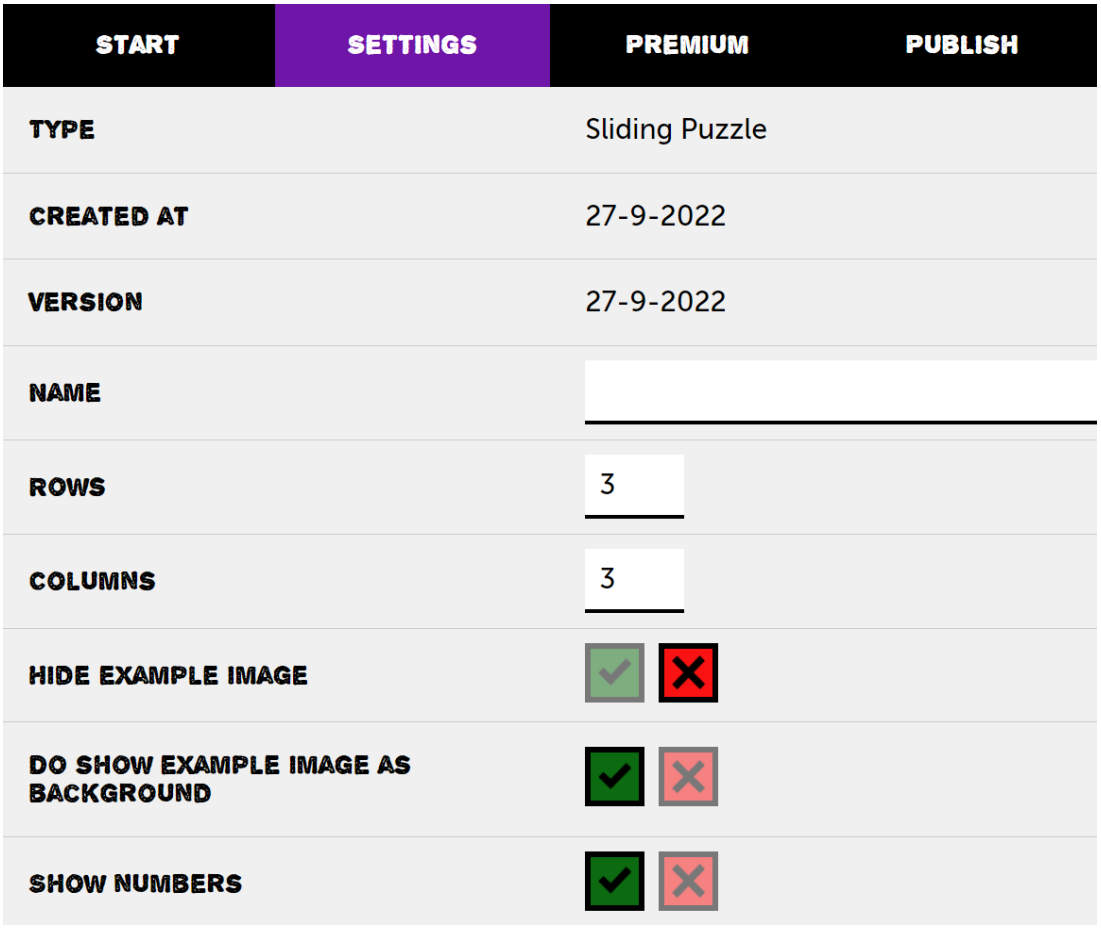
Match the Memory je alat koji omogućuje izradu igre memorije, a njegovo sučelje možemo vidjeti na slici 3. Uz pomoć ovog alata nastavnici mogu kreirati igru memorije tako što im dozvoljava da odaberu naslov svoje igre, broj parova kartica, sam sadržaj unutar kartica te razne vrste vizualnih promjena kao što su boja pozadine, boja teksta, pozadina samih kartica, veličina fonta itd.

[6] https://wordwall.net/about



Slika 3 – Match the Memory sučelje za izradu igre

Puzzel.org je alat uz pomoć kojeg se mogu kreirati razne vrste slagalica kao što su križaljka, osmosmjerka ili klizna slagalica. Ukoliko odaberemo opciju za kreiranje klizne slagalice možemo vidjeti sučelje kakvo je prikazano na slici 4. Alat nam omogućuje odabir broja redaka i stupaca unutar same slagalice, ali besplatna verzija alata ne dozvoljava nikakve promjene na dizajnu same igre.



Slika 4 – Puzzel.org sučelje za izradu klizne slagalice

3. JavaScript u kontekstu izrade igara

Prije nego što počnemo govoriti o JavaScriptu i njegovoj ulozi u kreaciji igara, trebamo vidjeti koji se jezici uglavnom koriste za izradu igara. C++ i C# su jezici koje većina igara koristi u svojoj izradi, a to je iz razloga što se upravo ti jezici nalaze u pozadini najpopularnijih programa za izradu igara[7], poput Unity[8] i Unreal Engine[9] programa koje koriste gotovo svi proizvođači igara u industriji.

Java je idući jezik koji se koristi za izradu igara, no on je i dizajniran s ciljem da se uz njegovu pomoć izrađuju aplikacije i igre, a sve što je potrebno znati o tom jeziku za izradu igara se nalazi u knjizi David Brackeena iz 2004. godine u kojoj objašnjava kako se kreiraju igre unutar Java jezika.

JavaScript s druge strane nije programski jezik koji se koristi za izradu igara, već za izradu mrežnog sadržaja te ga mnogi nazivaju jezikom internetske mreže, a upravo tako ga je opisao i Wilton (2004). Budući da se JavaScript može pronaći u pozadini mrežnih stranica, počinju se razvijati i mrežne igre izrađene u tom jeziku. „*S pojavom HTML5 standarda, razvoj mrežnih aplikacija postaje veoma popularan te se sve više i više kompanija prebacuje na razvoj igara unutar JavaScript jezika*“ (Egges, 2014: 1). Iz ovog citata možemo vidjeti da se od pojave HTML5 standarda počinje razmišljati o izradi igara i unutar JavaScript jezika.

No, da se vratimo na početak. Kada pomislimo na sam pojam „igra“, većina će, ovisno o dobi osobe, najprije pomisliti na razbibrigu ili gubljenje vremena. Ali, kako se nastava mijenja i napreduje, moramo se zapitati ima li mjesta za igre u edukaciji.

De Freitas u svom radu iz 2018. godine govori o napretku znanosti i našeg razumjevanja o igranju i samim igrama te dolazi do zaključka da igre mogu biti veoma

[7] Podatak s mrežnog izvora - https://theninehertz.com/blog/game-engines
[8] https://unity.com
[9] https://unrealengine.com

koristan alat u nastavi. To je iz razloga što se „koncept intrinzične motivacije nalazi u angažmanu korisnika koji se stvara kod igranja digitalnih igara“ (Habgood, 2011: 4).

Koliko god pokušali motivirati učenike na rad, ukoliko im gradivo ili način predavanja nisu zanimljivi, teško ćemo zadobiti njihovu pažnju, dok s druge strane, igre dobivaju pažnju mladih veoma jednostavno. Naime, upravo zbog te karakteristike da mogu privući pažnju učenika, računalne igre mogu biti veoma efikasan alat u nastavi koji će omogućiti zabavnije nastavno iskustvo te omogućiti pažljivije praćenje sadržaja. S ovim tvrdnjama se slažu i Noemí i Máximo koji u svom radu iz 2014. govore da su igre odličan alat s kojim postižemo ove ciljeve, te dodaju da se uz pomoć korištenja igara sadržaj prenosi atraktivno i efikasno.

Također možemo spomenuti i rad iz 2014. godine koji pokušava dokazati da je uživanje veoma bitno u nastavnom procesu. Autori Allen, Crosley, Snow i McNamara (2014) u svom istraživanju dolaze do zaključka da učenici postižu bolje rezultate u pisanju ukoliko uživaju u samom procesu učenja, a upravo taj užitak možemo pružiti učenicima koristeći razne vrste igara.

Budući da smo se sada bliže upoznali s igrama u kontekstu edukacije te samim JavaScript jezikom, vrijeme je da postavimo hipoteze ovog istraživanja. Iako postoje alati koje možemo iskoristiti za izradu edukativnih igara, za potrebe ovog istraživanja ćemo se poslužiti određenim programskim jezicima kako bi samostalno kreirali 3 tipa edukativnih igara, a to su kviz znanja, igra memorije te klizna slagalica. Budući da JavaScript nema puno mogućnosti za oblikovanje izgleda, za tu ćemo svrhu koristiti HTML i CSS, a hipoteza koju ćemo pokušati dokazati u ovom radu je ta da je JavaScript programski jezik dovoljno razvijen da se može koristiti za izradu svih potrebnih dijelova odabranih edukativnih igara.

4. Kreacija kviza znanja

Prva igra za koju ćemo provjeriti je li ju moguće izraditi uz pomoć JavaScript koda je vrsta igre koja se može koristiti za provjeru znanja, a to je **kviz znanja**. Pretpostavka da je ova igra najjednostavnija za izradu je i razlog zašto sam se odlučio da će ona prva biti kreirana u sklopu ovog rada. Za kreaciju ove igre ćemo koristiti najjednostavnije funkcije unutar JavaScript koda, a uz JavaScript ćemo koristiti i HTML i CSS jezike.

Za početak želimo da se pitanja i odgovori samog kviza nalaze unutar JavaScript koda te želimo da ovaj kod automatski generira kviz. Na taj način nećemo trebati ispisivati puno koda koji se ponavlja te ćemo u svakom trenutku moći dodavati nova i brisati postojeća pitanja.

Rad na dizajnu strukture kviza započinje s izradom najbitnijih dijelova unutar HTML jezika, a to su:

- dio koji sadržava kviz (Kodni blok 1 – linija 1),
- gumb za predavanje kviza (Kodni blok 1 – linija 2),
- dio koji prikazuje rezultate (Kodni blok 1 – linija 3).

```
1   <div id="quiz"></div>
2   <button id="submit">Predaj odgovore</button>
3   <div id="results"></div>
```

Kodni blok 1 – tri osnovna dijela koda u HTML-u

Elementima kviza smo dali identifikacije korištenjem `id` atributa te ćemo se na njih referirati unutar JavaScript koda kao što se vidi na primjeru u kodnom bloku 2.

```
1   const quizContainer = document.getElementById('quiz');
2   const resultsContainer = document.getElementById('results');
3   const submitButton = document.getElementById('submit');
```

Kodni blok 2 – Reference unutar JavaScript koda

Uz pomoć funkcije[10] `getElementById()` prikazane u kodnom bloku 2 možemo dobiti podatke koji se nalaze unutar kreiranih HTML elemenata te ćemo ih pohraniti u varijable definirane pomoću ključne riječi `const`. Sljedeće stvari koje će nam biti potrebne su način za izradu samog kviza te prikaz rezultata. Za sada možemo definirati funkcije, a njihovu funkcionalnost ćemo dodavati kako gradimo kviz.

```
1  function buildQuiz(){}
2
3  function showResults(){}
4
5  buildQuiz();
6
7  submitButton.addEventListener('click', showResults);
```

Kodni blok 3 – Funkcije u JavaScript kodu

Na kodnom bloku 3 imamo funkcije za izradu kviza (`function buildQuiz()`) i prikaz rezultata (`function showResults()`). Funkciju `buildQuiz()` ćemo pokrenuti odmah, a funkcija `showResults()` će se pokrenuti kada stisnemo na za to predviđen gumb.

Svakom kvizu su potrebna pitanje te je to iduća stvar koja se treba dodati. Za to ćemo koristiti objektne literale[11] koji će predstavljati individualna pitanja te niz koji će sadržavati sva pitanja od kojih se kviz sastoji. Korištenje niza za pitanja je odličan način za potrebe izrade ovog tipa igre budući da se i pitanja i odgovori nalaze na istom mjestu unutar koda.

[10] Funkcija je skup izraza koji izvršava zadatak ili izračunava vrijednost
[11] Objektni literali su popisi parova ime-vrijednost koje pišemo unutar vitičastih zagrada odvojenih zarezima.

```
const myQuestions = [
    {
      question: "Tko je izmislio JavaScript?",
      answers: {
        a: "Douglas Crockford",
        b: "Sheryl Sandberg",
        c: "Brendan Eich"
      },
      correctAnswer: "c"
    },
    {
      question: "Što od sljedećeg je upravitelj paketa unutar JavaScript jezika?",
      answers: {
        a: "Node.js",
        b: "TypeScript",
        c: "npm"
      },
      correctAnswer: "c"
    },
    {
      question: "Koji alat se može koristiti kako bi osigurali kvalitetu koda?",
      answers: {
        a: "Angular",
        b: "jQuery",
        c: "RequireJS",
        d: "ESLint"
      },
      correctAnswer: "d"
    }
  ];
```

Kodni blok 4 – Primjer pitanja za kviz znanja

U kodnom bloku 4 dana su tri primjer prema modelu za kreiranje pitanja:

(1) varijabli `question` zadaje se pitanje,

(2) varijabli `answers` zadaju se predloženi odgovori i

(3) varijabli `correctAnswer` zadaje se točan odgovor.

Na ovaj način se može dodati bilo koji broj pitanja, a ona će se u kvizu pojavljivati redosljedom kojim ih upisujemo u niz. Budući da sad imamo listu pitanja možemo ih prikazati na stranici.

```
1  function buildQuiz(){
2
3      const output = [];
4
5      myQuestions.forEach(
6        (currentQuestion, questionNumber) => {
7
8          const answers = [];
9
10         for(letter in currentQuestion.answers){
11
12           answers.push(
13             `<label>
14               <input type="radio" name="question${questionNumber}" value="${letter}">
15               ${letter} :
16               ${currentQuestion.answers[letter]}
17             </label>`
18           );
19         }
20
21         output.push(
22           `<div class="question"> ${currentQuestion.question} </div>
23           <div class="answers"> ${answers.join('')} </div>`
24         );
25       }
26     );
27
28     quizContainer.innerHTML = output.join('');
29   }
```

Kodni blok 5 – buildQuiz funkcija

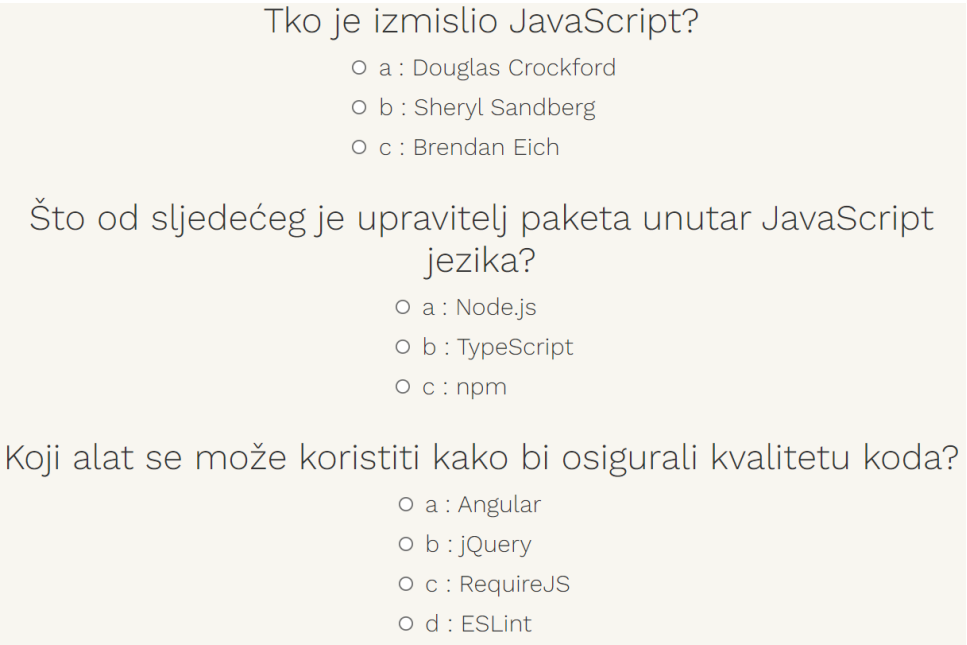
Na kodnom bloku 5 vidimo da su dodane linije (3..28) koda unutar funkcije `buildQuiz()`. Prvo je kreirana varijabla `output` koja sadrži sve izlaze koji se dobivaju u HTML kodu, kao što su pitanja i odgovori. Iduće se dodaje sadržaj u HTML elemente za svako pitanje, a za to je potrebna `forEach` petlja. Korištenjem takve petlje dobivamo trenutnu vrijednost, indeks i sami niz kao podatke na izlazu same petlje. Za potrebe ove igre su potrebni samo trenutna vrijednost i indeks, a njih smo unutar koda (Kodni blok 5 – linija 6) imenovali kao `currentQuestion` i `questionNumber`.

Za svako pitanje želimo generirati točne HTML oznake, a prvi korak je kreiranje niza koji sadrži listu svih mogućih odgovora. Nakon toga se koristi petlja kako bi se popunili mogući odgovori za trenutno pitanje. Za svaki izbor se kreira radio gumb (Kodni blok 5 – linija 13..17) koji se zatvara s `</label>` oznakom (Kodni blok 5 – linija 17). Taj dio je potreban ukoliko želimo da korisnik može kliknuti bilo gdje na odgovoru kako bi se

odabrao taj odgovor. Ukoliko ne dodamo taj element, korisnik bi morao kliknuti točno na radio gumb koji nije pretjerano pristupačan. Možemo vidjeti i da koristimo literale predloška[12] što su nizovi slični stringovima, ali omogućuju više funkcionalnosti. A upravo od tih mogućnosti ćemo iskoristiti sljedeće:

- višelinijske mogućnosti,
- mogućnost korištenja navodnika unutar navodnika budući da literali predloška koriste povratne oznake,
- interpolaciju nizova tako da možemo ubaciti JavaScript izraze direktno u niz.

Budući da koristimo literale predloška, unutar koda možemo ubaciti više elemenata koji zahtjevaju korištenje duplih navodnika, što nam omogućuje da unutar JavaScript koda ubacujemo HTML kod. To nam omogućuje kreaciju podjele za pitanja i podjele za odgovore unutar istog dijela koda. Pomoću `join` izraza lista odgovora se stavlja u jedan niz kao izlaz za `answers` element. Na ovaj se način generira HTML kod za svako pojedinačno pitanje te ih sada možemo spojiti i prikazati na stranici (Slika 5).



Slika 5 – Prikaz pitanja kviza znanja

[12] Literali predloška su literali razgraničeni znakovima povratne kvačice (`), dopuštajući nizove s više redaka, interpolaciju nizova s ugrađenim izrazima i posebne konstrukcije koje se nazivaju označeni predlošci.

Kao što vidimo na slici 5, dodana pitanja su prikazana redosljedom kakvim su i napisani u samom kodu. Takav prikaz možemo povezati i s logikom samog JavaScript jezika koji kod čita odozgo prema dolje, što znači da je potrebno paziti da je struktura koda dobro napisana. Budući da JavaScript čita kod odozgo prema dolje, ne možemo pozvati nešto što još nije definirano.

Nakon što smo završili s funkcijom za izradu kviza, započinjemo s opisom funkcije `showResults()` kojoj je zadatak proći kroz odgovore, pregledati ih i prikazati ukupan rezultat rješavanja kviza (Kodni blok 6).

```
1  function showResults(){
2
3      const answerContainers = quizContainer.querySelectorAll('.answers');
4
5      let numCorrect = 0;
6
7      myQuestions.forEach( (currentQuestion, questionNumber) => {
8
9        const answerContainer = answerContainers[questionNumber];
10       const selector = `input[name=question${questionNumber}]:checked`;
11       const userAnswer = (answerContainer.querySelector(selector) || {}).value;
12
13       if(userAnswer === currentQuestion.correctAnswer){
14
15         numCorrect++;
16
17         answerContainers[questionNumber].style.color = 'lightgreen';
18       }
19
20       else{
21
22         answerContainers[questionNumber].style.color = 'red';
23       }
24     });
25
26     resultsContainer.innerHTML = `${numCorrect} out of ${myQuestions.length}`;
27   }
```

Kodni blok 6 – showResults funkcija

Na kodnom bloku 6 vidimo da se prvo odabiru svi odgovori spremljeni u kontejnere pomoću `querySelectorAll()` funkcije (Kodni blok 8 – linija 3) te se zatim kreiraju varijable koje prate korisnikov trenutni odgovor i broj točnih odgovora. Zatim je potrebno izraditi petlju koja prolazi kroz sva pitanja i provjerava rezultate.

Linije 7..24 na kodnom bloku 6 prikazuju petlju koja radi sljedeće:

- pronalazi odabrani odgovor,
- brine se što se dogodi ako je odgovor točan,
- brine se što se dogodi ako je odgovor netočan.

Možemo i pobliže pogledati na koji način se traže odabrani odgovori. Na linijama 9..11 na kodnom bloku 6 vidimo da se prvo provjerava kontejner s odgovorom za trenutno pitanje. Nakon toga je definiran CSS selektor koji nam dopušta da pronađemo koji je *radio gumb* odabran. Zatim se koristi `querySelector` funkcija da pronađemo taj CSS selektor u prije definiranom `answerContainer` kontejneru. Uglavnom, ovo znači da će se pronaći koji je radio gumb za odgovor označen. Na kraju se dobiva vrijednost koristeći `.value` svojstvo. Problem koji se sada može pojaviti je da korisnik ostavi odgovor prazan te bi u tom slučaju korištenje `.value` svojstva uzrokovalo pogrešku zato što se ne može dobiti vrijednost nečega čega nema. Da bi izbjegli taj problem dodali smo `||` što znači „ili“ te smo dodali i vitičaste zagrade `{}` koje označavaju prazan objekt. Sada će ova funkcija raditi sljedećom logikom:

- dobavi podatak o odabranom odgovoru ili ako odgovor ne postoji iskoristi prazan objekt,
- Zatim uzmi vrijednost dobivenu u prvoj izjavi.

Kao rezultat vrijednost će biti ili korisnikov odgovor ili nedefiniran što znači da korisnik može preskočiti pitanje bez uzrokovanja rušenja samog kviza. Sljedeće izjave unutar petlje koja provjerava odgovore omogućuje da se nosimo s točnim i netočnim odgovorima.

Na kodnom bloku 6 (Linija 13..22) vidimo da ukoliko se korisnikov odgovor jednači s točnim odabirom povećavamo broj točnih odgovora za jedan i bojamo odabir zeleno, a ukoliko je odgovor netočan ili prazan bojamo odabir u crveno.

Posljednja linija koda u kodnom bloku 6 (Linija 26) prikazuje što radimo nakon što završimo s provjerom odgovora, a to je mogućnost prikazivanja na koliko je pitanja korisnik točno odgovorio.

Sada imamo funkcionalan kviz kreiran unutar JavaScript jezika te se sada mogu dodati i neka naprednija svojstva. Jedno od tih svojstava je prikazivanje jednog pitanja u jednom trenutku, dakle ne želimo sva pitanja odjednom imati na ekranu već želimo da korisnik odgovara na pitanja jedno po jedno. Za to je potreban način za prikazivanje i sakrivanje pitanja te gumbi koji će služiti za navigaciju kvizom. Trebamo ažurirati kod, a započet ćemo s HTML dijelom.

```
<div class="quiz-container">
  <div id="quiz"></div>
</div>
<button id="previous">Prethodno pitanje</button>
<button id="next">Sljedeće pitanje</button>
<button id="submit">Predaj odgovore</button>
<div id="results"></div>
```

Kodni blok 7 – Ažurirani HTML kod

Na kodnom bloku 7 vidimo da je većina koda ista kao i prije, ali su dodani navigacijski gumbi uz pomoć `<button>` elementa i kontejner kviza `quiz-container` pomaže kod pozicioniranja pitanja tako što će ih pretvoriti u slojeve slojeve što omogućuje sakrivanje i prikazivanje pitanja po potrebi.

```
output.push(
  `<div class="slide">
    <div class="question"> ${currentQuestion.question} </div>
    <div class="answers"> ${answers.join("")} </div>
  </div>`
);
```

Kodni blok 8 – Klasa slide unutar buildQuiz funkcije

Sljedeće smo unutar `buildQuiz()` funkcije dodali `<div>` element s klasom `slide` koja će sadržavati pitanja i kontejner odgovora koji smo kreirali kao što je prikazano na kodnom bloku 8.

Zatim je iskorišten CSS kod za pozicioniranje kako bi napravili da pitanja stoje kao slojevi jedan na drugom.

```
1  .slide{
2      position: absolute;
3      left: 0px;
4      top: 0px;
5      width: 100%;
6      z-index: 1;
7      opacity: 0;
8      transition: opacity 0.5s;
9    }
10   .active-slide{
11     opacity: 1;
12     z-index: 2;
13   }
14   .quiz-container{
15     position: relative;
16     height: 200px;
17     margin-top: 40px;
18   }
```

Kodni blok 9 – CSS kod za pojavljivanje i nestajanje pitanja

Za naše potrebe, koristimo z indekse i prijelaze prozirnosi (kodni blok 9 – linija 6...8) kako bi pitanja unutar kviza postepeno nestajala i postepeno se stvarala kao što je prikazano u kodnom bloku 9.

Sada je potrebno dodati JavaScript kod kako bi ova zamisao zapravo i funkcionirala. Kao što je već napisano, redosljed je bitan tako da se mora pripaziti gdje se novi kod dodaje. Možemo započeti dodavanjem varijabli koje će pohranjivati podatke o gumbima za navigaciju kako bi mogli pratiti na kojem smo pitanju. Kod se dodaje nakon što smo pozvali `buildQuiz()` funkciju.

```
1  const previousButton = document.getElementById("previous");
2  const nextButton = document.getElementById("next");
3  const slides = document.querySelectorAll(".slide");
4  let currentSlide = 0;
```

Kodni blok 10 – Funkcionalnost gumba za navigaciju

Sljedeća stvar nakon što se omogućila navigacija pomoću koda prikazanog na kodnom bloku 10 je dodavanje funkcije koja prikazuje pitanje. Ovu funkciju se dodaje nakon već kreiranih funkcija.

```
1  function showSlide(n) {
2      slides[currentSlide].classList.remove('active-slide');
3      slides[n].classList.add('active-slide');
4      currentSlide = n;
5      if(currentSlide === 0){
6        previousButton.style.display = 'none';
7      }
8      else{
9        previousButton.style.display = 'inline-block';
10     }
11     if(currentSlide === slides.length-1){
12       nextButton.style.display = 'none';
13       submitButton.style.display = 'inline-block';
14     }
15     else{
16       nextButton.style.display = 'inline-block';
17       submitButton.style.display = 'none';
18     }
19   }
```

Kodni blok 11 – Kod za navigaciju kvizom

Prve tri linije koda prikazanog na kodnom bloku 11 rade sljedeće:

- sakrivamo trenutno pitanje uklanjanjem `active-slide` klase,
- prikazujemo novo pitanje dodavanjem `active-slide` klase,
- ažuriramo trenutni broj pitanja.

Linije koje dolaze nakon (Kodni blok 11 – linija 5..18) predstavljaju korištenje logike koju možemo opisati sljedećim pseudokodom:

- IF na prvom pitanju THEN sakrij gumb za prethodno pitanje ELSE prikaži taj gumb,

- IF na posljednjem pitanju THEN (sakrij gumb za iduće pitanje AND prikaži gumb za predavanje kviza) ELSE (prikaži gumb za iduće pitanje AND sakrij gumb za predavanje kviza).

Nakon što je kreirana funkcija za prikazivanje pitanja, odmah se može pozvati `showSlide(0)` kako bi se prikazalo prvo pitanje.

```
1   //Nakon koda za navigaciju
2
3   showSlide(currentSlide);
```

Kodni blok 12 – Pozivanje showSlide funkcije

Nakon što smo pozvali funkciju `showSlide()` kao što je prikazano u kodnom bloku 12, želimo kreirati funkcije koje će omogućiti gumbima za navigaciju da zapravo i funkcioniraju.

```
1   function showNextSlide() {
2       showSlide(currentSlide + 1);
3   }
4
5   function showPreviousSlide() {
6       showSlide(currentSlide - 1);
7   }
```

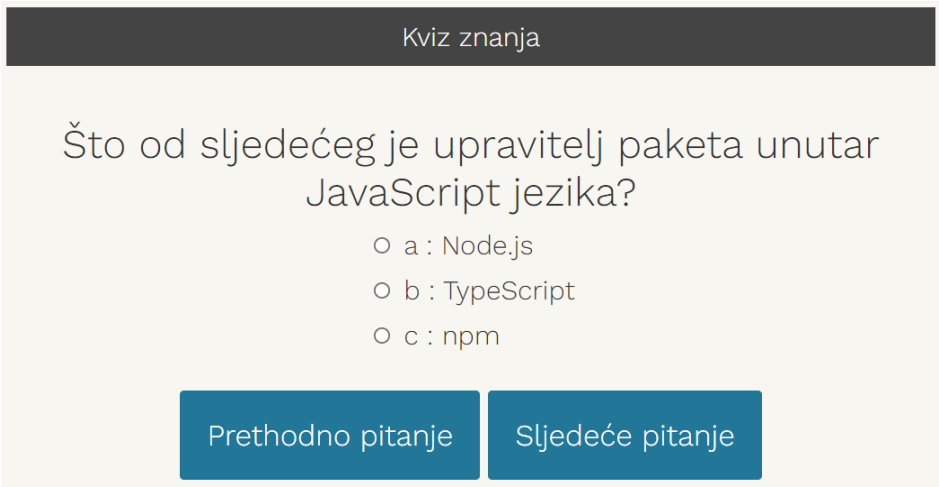
Kodni blok 13 – showSlide funkcija

U kodnom bloku 13 možemo vidjeti da se koristi `showSlide()` funkcija kako bi omogućili gumbima za navigaciju da mogu prikazati sljedeće i prethodno pitanje tako što povećavamo ili smanjujemo broj trenutnog pitanja. Za kraj se gumbi za navigaciju trebaju spojiti s ovim funkcijama, a to dolazi na kraju koda.

```
1   previousButton.addEventListener("click", showPreviousSlide);
2   nextButton.addEventListener("click", showNextSlide);
```

Kodni blok 14 – Navigacija uz pomoć funkcija

Zaključno s kodom prikazanim u kodnom bloku 14 imamo jednostavan kviz kreiran unutar JavaScript jezika.

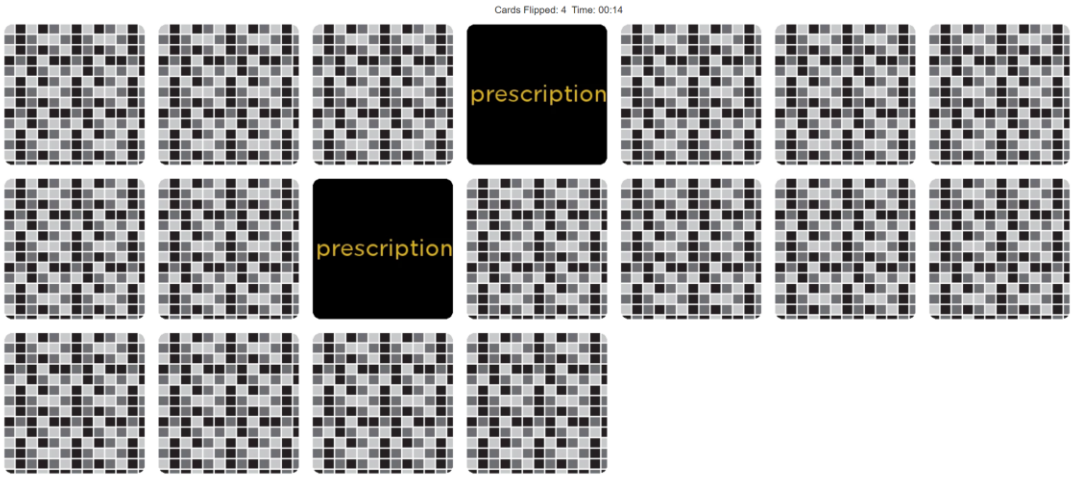


Slika 6 – Prikaz kviza znanja

Ukoliko pogledamo kako kviz znanja izgleda, vidimo da sve radi kako je predviđeno, na prvom pitanju nećemo imati gumb za prethodno pitanje te na zadnjem pitanju imamo gumb za predaju odgovora, a ukoliko smo na pitanju koje nije niti prvo niti zadnje imamo gumbe za prethodno i sljedeće pitanje kao što se može vidjeti i na slici 6. Što se tiče ovog tipa igre mislim da je jasno da se kviz znanja i svi potrebni elementi mogu kreirati unutar JavaScript jezika. Sada idemo na malo kompliciraniju vrstu igre: igra memorije.

5. Kreacija igre memorije

Druga vrsta igre za koju ćemo provjeriti je li ju moguće izraditi uz korištenje JavaScript programskog jezika je igra koja je nama svima poznata te se odnosi na poboljšanje pamćenja. CARNET kao jednu takvu igru za upotrebu u edukativne svrhe nudi *„Match the Memory“*. Sve što trebamo znati o ovoj igri možemo pronaći u knjizi Zwicka i Patersona (1993). Za početak trebamo saznati od kojih dijelova se sama igra sastoji kako bi smo mogli procijeniti za koje dijelove se može koristiti JavaScript, a za koje ne.



Slika 7 – Match the Memory

Slika 7 pokazuje kako igra izgleda unutar internetskog preglednika, a na samoj slici vidimo da se igra sastoji od broja poteza, vremena koje je proteklo za vrijeme igranja te od kartica koje trebamo spojiti. Kada su dvije povezane kartice otvorene, one ostaju otvorene do kraja igre, a ukoliko odaberemo pogrešno one se ponovno okreću te se igra završava kada točno povežemo sve kartice. Sad kada znamo koje dijelove trebamo kreirati, možemo započeti s izradom same igre, a za početak ćemo izraditi HTML dokument unutar kojeg ćemo pozvati kod kreiran u JavaScript programskom jeziku.

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Igra sjećanja u JavaScript jeziku</title>

    <link rel="stylesheet" href="assets/styles.css" />
    <script src="assets/game.js" defer></script>
</head>

<body>
    <div class="game">
        <div class="controls">
            <button>Započni</button>
            <div class="stats">
                <div class="moves">0 poteza</div>
                <div class="timer">Vrijeme: 0 sec</div>
            </div>
        </div>
        <div class="board-container">
            <div class="board" data-dimension="4"></div>
            <div class="win">Bravo!</div>
        </div>
    </div>
</body>

</html>
```

Kodni blok 15 – HTML kod za igru memorije

Sve što nam je potrebno za funkcionalnost igre će se nalaziti unutar `.game` kontejnera kao što se može vidjeti na kodnom bloku 15 (Linija 15..27). Prikazivat ćemo broj poteza gdje će se svaki klik mišom računati kao jedan potez te ćemo prikazivati koliko vremena je proteklo od početka igre. Element `.board` ima atribut `.data-dimension` koji ćemo koristiti za generiranje ploče te taj atribut možemo promjeniti s lakoćom ukoliko želimo veću ploču, a za svrhu ovog projekta ćemo koristiti ploču veličine četiri puta četiri. Također ćemo korsititi dva sredstva koja ćemo povezati s dokumentom, prva je tablica stilova koja nas povezuje sa CSS kodom koji koristimo za oblikovanje, a drugo sredstvo predstavlja ulaznu točku za igru.

U ovom dijelu ćemo se fokusirati na izgled same ploče. Budući da se ovaj rad fokusira na JavaScript pojasnit ćemo samo najbitnije dijelove CSS koda. Za početak ćemo pozvati prilagođeni font koji možemo pronaći na *Google Fonts* stranici

```
@font-face {
    font-family: Fredoka;
    src: url(./FredokaOne-Regular.ttf);
}

font-family: Fredoka;
```

Kodni blok 16 – Stiliziranje fonta igre

Kodni blok 16 prikazuje kako se u CSS poziva prilagođeni font unutar CSS jezika. Prvo upisujemo koja je obitelj fonta, u ovom slučaju to je *Fredoka*, a nakon toga dodajemo izvor koji upisujemo unutar `src` elementa.

```
.game {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
}
```

Kodni blok 17 – Pozicija ploče za igru

Uz font koji smo odabrali želimo i da cijela ploča bude na sredini ekrana. To možemo postići pomoću apsolutnog pozicioniranja i korištenja transformiranja. Kod je prikazan na kodnom bloku 17 i on nam omogućuje da se ploča za igru uvijek nalazi na sredini ekrana neovisno o rezoluciji samog prozora unutar kojeg se nalazi korištenjem `position: absolute;` parametra.

```css
button {
    cursor: pointer;
}

.disabled {
    color: #757575;
}
```

Kodni blok 18 – Kursor prije i nakon početka igre

Također želimo da se kursor promjeni kada se nalazi na gumbu za početak igre. Kodni blok 18 prikazuje kod kojim to postižemo kako bi korisnicima dali do znanja da se na taj gumb može kliknuti, a u tu svrhu koristimo `cursor: pointer;` svojstvo. Također želimo da se klik na taj gumb onemogući za vrijeme igranja, te smo zato kreirali i stanje koje možemo nazvati `.disabled` te ćemo to stanje pozvati unutar JavaScript koda.

Zadnja stvar koju je potrebno napraviti unutar CSS dijela ovog programa je omogućiti okretanje kartica. Prvo ćemo vidjeti što se sve generira unutar klase `.board` kako bi razumjeli koji će sve elementi biti potrebni za rad.

```html
<div class="board" data-dimension="4">
    <div class="card">
        <div class="card-front"></div>
    <div class="card-back"></div>
    </div>
</div>
```

Kodni blok 19 – Elementi kartica za igru

Svaka kartica će imati 3 `<div>` elementa kao što je prikazano na kodnom bloku 19, a to su:

- kontejner u koji dodajemo dimenzije koji je nazvan `card`,
- element za prednju stranu kartice – prazna strana koju vidimo kada pokrenemo igru,

- element za stražnju stranu kartice – strana koja je skrivena dok ne okrenemo karticu te se na njoj nalazi ono što trebamo spojiti.

Za okretanje ćemo koristiti svojstvo `transform`. Za element `card` ćemo ovdje dodati neke dimenzije.

```
1  .card {
2      position: relative;
3      width: 100px;
4      height: 100px;
5      cursor: pointer;
6  }
```

Kodni blok 20 – Dimenzije kartica

U kodnom bloku 20 vidimo prikaz dodanih dimenzija, pozicija, visina i širina te ono što je bitno kako bi prikazali da su kartice interaktivne, a to je promjena kursora što dobivamo koriteći `cursor: pointer` parametar.

```
1  card-front,
2  .card-back {
3      position: absolute;
4      border-radius: 5px;
5      width: 100%;
6      height: 100%;
7      background: #282A3A;
8      transition: transform .6s cubic-bezier(0.4, 0.0, 0.2, 1);
9      backface-visibility: hidden;
10 }
```

Kodni blok 21 – Kod za animaciju okretanja kartica

Najbitnije stvari u ovom dijelu koda prikazanom na kodnom bloku 21 su zadnje dvije linije odnosno pravila. Koristeći svojstvo `transition` na svojstvu `transform` kreiramo animaciju za okretanje kartica (Kodni blok 21 – linija 8). Dodavanje `cubic-bezier` svojstva ublažava animaciju kako bi izgledala ugodnije tako što nam omogućuje da postupno ubrzavamo i usporavamo animaciju. Drugo svojstvo koje je bitno u ovom dijelu koda je `backface-visibility` (Kodni blok 21 – linija 9) koje nam omogućuje da sakrijemo sadržaj koji se nalazi na drugoj strani kartice.

Kako bi točno rotirali kartice, u CSS se dodaje kod prikazan u kodnom bloku 22.

```
.card-back {
    font-size: 28pt;
    text-align: center;
    line-height: 100px;
    background: #FDF8E6;
    transform: rotateY(180deg) rotateZ(50deg);
    user-select: none;
}

.card.flipped .card-front {
    transform: rotateY(180deg) rotateZ(50deg);
}

.card.flipped .card-back {
    transform: rotateY(0) rotateZ(0);
}
```

Kodni blok 22 – Rotacija kartica

CSS kod prikazan u kodnom bloku 22 omogućuje rotaciju kartice 180 stupnjeva što znači da ćemo poslije rotacije vidjeti stranu koju trebamo spojiti, a sakrit ćemo stranu na kojoj nemamo ništa. Ukoliko su odabrane dvije pogrešne kartice, odnosno one koje se ne spajaju, ponovno se okreću, a ovo pravilo se koristi i za cijelu ploču kada završimo sa igrom.

Uz pomoć HTML i CSS programskih jezika je napravljen raspored i stil igre, ali ta dva jezika nam ne omogućuju nikakvu funkcionalnost same igre. Za početak JavaScript koda za igru ćemo definirati određene selektore[13] i stanja same igre uz pomoć koda prikazanog na kodnom bloku 23.

[13] Selektori se koriste kako bi pristupili određenim HTML elementima koristeći njegove oznake

```
1  const selectors = {
2      boardContainer: document.querySelector('.board-container'),
3      board: docum ent.querySelector('.board'),
4      moves: document.querySelector('.moves'),
5      timer: document.querySelector('.timer'),
6      start: document.querySelector('button'),
7      win: document.querySelector('.win')
8  }
9
10 const state = {
11     gameStarted: false,
12     flippedCards: 0,
13     totalFlips: 0,
14     totalTime: 0,
15     loop: null
16 }
```

Kodni blok 23 – Selektori i stanja igre

Dvije konstante (Kodni blok 23 – linija 1 i linija 10) koje smo dodali u kod nam omogućuju da pristupimo određenim selektorima više puta, a što se tiče samog stanja igre, trebamo pratiti 5 različitih vrijednosti:

- booleova vrijednost koja prati je li igra započela ili ne,
- `flippedCards` vrijednost će pratiti koliko puta smo okrenuli karte, samo dvije karte se mogu okrenuti u isto vrijeme te ukoliko se podudaraju ostaju okrenute do završetka igre, a ukoliko se ne podudaraju vraćaju se u originalno stanje,
- `totalFlips` vrijednost prati koliki je ukupan broj okretaja tijekom igranja,
- `totalTime` vrijednost prati koliko je vremena prošlo od početka igre,
- *`loop`* vrijednost se koristi za petlju same igre te ažurira vrijeme igre nakon svake sekunde.

Generiranje ploče je sljedeći korak koji ćemo dodati u ovu igru. Taj korak se temelji na atributu `data-dimension` koji je prethodno definiran. Za to se koristiti nova funkcija koju smo nazvali `generateGame`, koja će se pozvati na kraju JavaScript dokumenta.

```
1  const generateGame = () => {
2      const dimensions = selectors.board.getAttribute('data-dimension')
3
4      if (dimensions % 2 !== 0) {
5          throw new Error("The dimension of the board must be an even number.")
6      }
7
8      const emojis = ['🥔', '🍒', '🍆', '🍑', '🥕', '🍇', '🍉', '🍌', '🥭', '🍍']
9      const picks = pickRandom(emojis, (dimensions * dimensions) / 2)
10     const items = shuffle([...picks, ...picks])
11     const cards = `
12         <div class="board" style="grid-template-columns: repeat(${dimensions}, auto)">
13             ${items.map(item => `
14                 <div class="card">
15                     <div class="card-front"></div>
16                     <div class="card-back">${item}</div>
17                 </div>
18             `).join('')}
19        </div>
20     `
21
22     const parser = new DOMParser().parseFromString(cards, 'text/html')
23
24     selectors.board.replaceWith(parser.querySelector('.board'))
25 }
```

Kodni blok 24 – Generiranje ploče

Kao prvi korak unutar funkcije prikazane na kodnom bloku 24 želimo provjeriti da je dimenzija koju smo odabrali paran broj (Kodni blok 24 – linija 4), to najjednostavnije možemo provjeriti koristeći ostatak. Ova varijabla se može koristiti kasnije za dinamičko postavljanje stila ploče za igru unutar ovog predloška. Kako bi to dodali u DOM[14] (model objekta dokumenta) koristimo `domParser`. Za samo generiranje nasumičnih stavki koristimo dvije funkcije:

- `pickRandom` funkcija omogućuje da se nasumično odabiru stavke iz odabranog niza. Želimo izabrati emotikone iz kreiranog niza te želimo da taj broj bude polovica broja dimenzije. Na primjer, ukoliko nam je za dimenziju odabran broj 4, generirat će se ploča sa 16 kartica odnosno dimenzije 4x4, a

[14] DOM (Document Object Model) – standard kojim se pristupa dokumentima

to znači da trebamo odabrati 8 stavki iz niza jer nam trebaju parovi za svaki emotikon,

- `shuffle` funkciju koristimo za miješanje niza kako bi nam kartice uvijek bila nasumično poredane.

```
1  const pickRandom = (array, items) => {
2      const clonedArray = [...array]
3      const randomPicks = []
4
5      for (let index = 0; index < items; index++) {
6          const randomIndex = Math.floor(Math.random() * clonedArray.length)
7
8          randomPicks.push(clonedArray[randomIndex])
9          clonedArray.splice(randomIndex, 1)
10     }
11
12     return randomPicks
13 }
```

Kodni blok 25 – Funkcija za nasumičan odabir

`pickRandom` funkcija nakon kloniranja originalnog niza ponovno prolazi kroz niz onoliko puta koliko je potrebno da popunimo ploču te ih stavlja na nasumično odabranu poziciju na samoj ploči te vraća *`randomPicks`* na kraju funkcije kao što možemo vidjeti na kodnom bloku 25 (Linija 12). Zatim se prosljeđuje u *`shuffle`* funkciju koja koristi Fisher-Yates algoritam za miješanje koji je opisan u njihovoj knjizi (Fisher i Yates, 1953). Prvo trebamo vidjeti pseudo kod samog algoritma kako bi saznali kako ga možemo implementirati unutar JavaScript jezika.

```
1  -- To shuffle an array a of n elements (indices 0..n-1):
2  for i from n−1 downto 1 do
3      j ← random integer such that 0 ≤ j ≤ i
4      exchange a[j] and a[i]
```

Slika 8 – Fisher-Yates algoritam za miješanje

Na algoritmu prikazanom na slici 8 vidimo da su nam potrebna tri ključna koraka:

- od `i` do `<duljina niza> −1`, sve dolje do `1`, želimo:

- kreirati nasumičan cijeli broj koji je veći od 0, ali manji od indeksa. Taj broj ćemo označiti sa `j`,
- Zatim ćemo zamjeniti nasumični indeks sa `i` tako da je `niz [i] = niz[j]` i `niz[j] = nit[i]`

Ovo znači da `shuffle` funkcija treba izgledati kako je prikazana na kodnom bloku 26.

```
const shuffle = array => {
    const clonedArray = [...array]

    for (let index = clonedArray.length - 1; index > 0; index--) {
        const randomIndex = Math.floor(Math.random() * (index + 1))
        const original = clonedArray[index]

        clonedArray[index] = clonedArray[randomIndex]
        clonedArray[randomIndex] = original
    }

    return clonedArray
}
```

Kodni blok 26 – Funkcija za miješanje kartica

Unutar koda prikazanog u kodnom bloku 26 koristimo silaznu `for` petlju unutar koje kreiramo nasumičan indeks koristeći `Math.random` i `Math.floor` funkcije te nam to omogućuje da zamjenimo pozicije dviju stavki unutar niza. Ovo nam omogućuje da se ploča generira s nasumično poredanim emotikonima svaki put, ali i dalje ne možemo okretati kartice te trebamo dodati potrebne slušatelje događaja[15] unutar koda.

Potrebna su nam 2 različita slušatelja događaja, jedan za kartice i jedan za gumb za početak igre. Dodat ćemo novu funkciju te ćemo ju pozvati nakon generiranja same igre.

[15] Slušatelj događaja je objekt koji obrađuje događaj

```
const attachEventListeners = () => {
    document.addEventListener('click', event => {
        const eventTarget = event.target
        const eventParent = eventTarget.parentElement

        if (eventTarget.className.includes('card') && !eventParent.className.includes('flipped')) {
            flipCard(eventParent)
        } else if (eventTarget.nodeName === 'BUTTON' && !eventTarget.className.includes('disabled')) {
            startGame()
        }
    })
}

generateGame()
attachEventListeners()
```

Kodni blok 27 – Slušatelji događaja

Budući da se kartice dinamično dodaju u DOM želimo delegirati slušatelje događaja umjesto izravnog dodavanja na kartice kao što je prikazano u kodnom bloku 27. Delegacija događaja unutar JavaScripta se koristi za odgovaranje na korisničke događaje preko jednog roditelja, a ne preko svakog podređenog čvora što znači da možemo vezati jedan rukovoditelj događaja na zajednički element koji će zatim moći rukovati s bilo kojim podređenim događajem. Iz tog razloga slušatelja događaja moramo vezati za dokument te zatim ovisno o nazivu klase, možemo birati koju funkciju želimo pozvati:

- `flipCard` funkciju spajamo sa `.card` te ju pozivamo samo kada kartica još nije okrenuta,
- `startGame` funkciju pozivamo ako je gumb za početak pritisnut a nije onemogućen.

Prije nego što omogućimo samo okretanje kartica trebamo pogledati kako igra započinje. Za to će nam biti potrebna nova funkcija koju ćemo nazvati `startGame` te ćemo preko nje postaviti točno stanje igre.

```
1  const startGame = () => {
2      state.gameStarted = true
3      selectors.start.classList.add('disabled')
4
5      state.loop = setInterval(() => {
6          state.totalTime++
7
8          selectors.moves.innerText = `${state.totalFlips} moves`
9          selectors.timer.innerText = `time: ${state.totalTime} sec`
10     }, 1000)
11 }
```

Kodni blok 28 – Pokretanje igre

Prvo ćemo postaviti `gameStarted` zastavicu na `true` vrijednost kao što je prikazano u kodnom bloku 28 (Linija 2). Ovu zastavicu možemo koristiti i u drugim dijelovima ove aplikacije kako bi vidjeli da je igra pokrenuta. Također želimo onemogućiti gumb za početak igre da spriječimo pokretanje igre nakon što je već pokrenuta (Kodni blok 28 – linija 3). Također će nam trebati i petlja koja će se osvježiti svaku sekundu kako bi ažurirala broj okretaja kartica i proteklo vrijeme od početka igre. Valja napomenuti da ćemo za ovaj interval dodijeliti `state.loop` zastavicu kako bi ju mogli očistiti nakon što je igra završena.

Za okretanje kartica potrebno nam je malo više kodiranja, no ništa pretjerano zahtjevno.

```
const flipCard = card => {
    state.flippedCards++
    state.totalFlips++

    if (!state.gameStarted) {
        startGame()
    }

    if (state.flippedCards <= 2) {
        card.classList.add('flipped')
    }

    if (state.flippedCards === 2) {
        const flippedCards = document.querySelectorAll('.flipped:not(.matched)')

        if (flippedCards[0].innerText === flippedCards[1].innerText) {
            flippedCards[0].classList.add('matched')
            flippedCards[1].classList.add('matched')
        }

        setTimeout(() => {
            flipBackCards()
        }, 1000)
    }
}
```

Kodni blok 29 – Okretanje kartica

Prvo trebamo ažurirati stanje. Svaki put kada dođe do okretaja kartice moramo promjeniti i `flippedCards` i `totalFlips` stanja kako je prikazano na kodnom bloku 29. Zatim treba definirati individualne `if` izjave:

- prvo želimo provjeriti je li igra započeta. Ukoliko nije pozivamo `startGame` funkciju. Iako ovo nije obvezno, također je omogućeno započinjanje igre klikom na jednu od kartica.
- Zatim želimo provjeriti je li `flippedCards` manje ili jednako 2. Ne želimo dopustiti da igrač više od dvije kartice okrene u isto vrijeme. Ukoliko više od dvije kartice nisu okrenute možemo ih okrenuti dodavanjem `.flipped` klase `card` elementu koji šaljemo funkciji.
- Za kraj želimo provjeriti jesu li točno dvije kartice okrenute. U ovom slučaju moramo napraviti dvije različite stvari budući da se kartice mogu podudarati ili razlikovati:

- prvo ćemo dobiti `.flipped` kartice koje se još ne podudaraju a zatim treba provjeriti vrijednost unutar prve kartice te ju usporediti s vrijednosti druge kartice koristeći `innerText`. Ukoliko se podudaraju spajamo ih dodavajući `.matched` klasu tim karticama.
- Druga stvar koju želimo je da se sve okrenute kartice vrate u prvobitno stanje nakon jedne sekunde. Također dodajemo i funkciju prikazanu u kodnom bloku 30.

```
const flipBackCards = () => {
    document.querySelectorAll('.card:not(.matched)').forEach(card => {
        card.classList.remove('flipped')
    })

    state.flippedCards = 0
}
```

Kodni blok 30 – Okretanje kartica u izvorno stanje

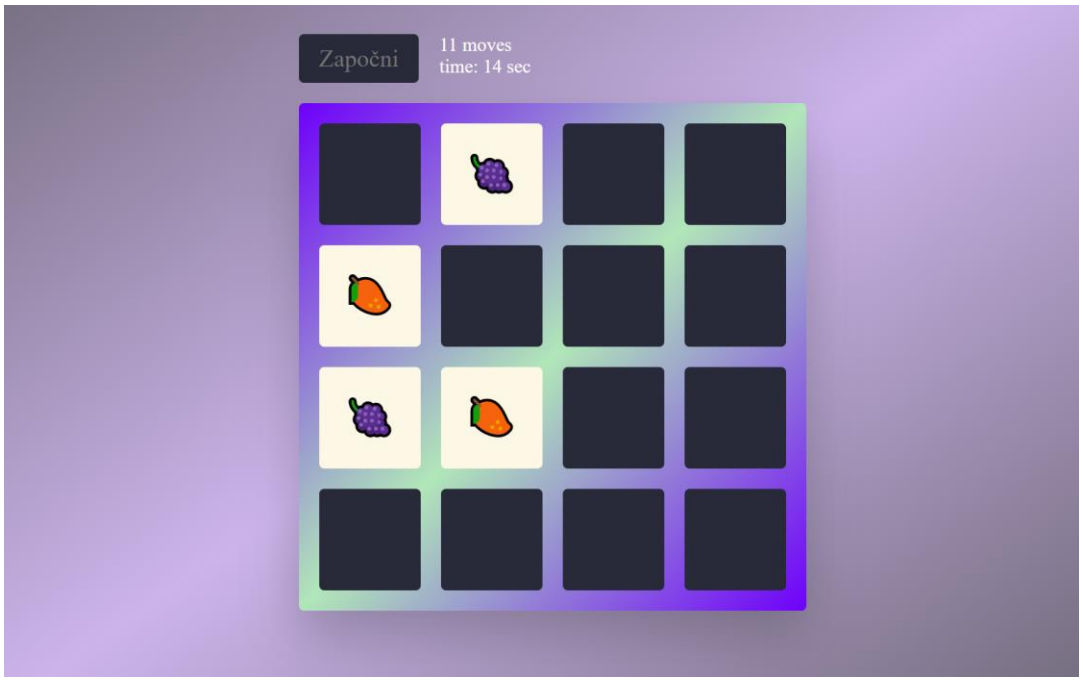
Funkcija `flipBackCards` koju vidimo u kodnom bloku 30 nam omogućuje da sve kartice koje se okrenu a ne podudaraju ponovno okrenu licem prema dolje tako što ćemo im maknuti `.flipped` klasu i vratiti `flippedCards` stanje na vrijednost 0. To će nam omogućiti da možemo nastaviti s okretanjem dvije kartice u isto vrijeme.

Sve što nam je još ostalo za napraviti je kreirati stanje za pobjedu igre. Za ovo ćemo proširiti `flipCard` funkciju s `if` izjavom koju možemo vidjeti u kodnom bloku 31.

```
if (!document.querySelectorAll('.card:not(.flipped)').length) {
    setTimeout(() => {
        selectors.boardContainer.classList.add('flipped')
        selectors.win.innerHTML = `
            <span class="win-text">
                You won!<br />
                with <span class="highlight">${state.totalFlips}</span> moves<br />
                under <span class="highlight">${state.totalTime}</span> seconds
            </span>
        `

        clearInterval(state.loop)
    }, 1000)
}
```

Kodni blok 31 – Završetak igre

Ovako ćemo provjeriti jesu li sve kartice okrenute, te ukoliko nema kartica koje nisu okrenute, znamo da je igra završena pa možemo pokazati pobjedničko stanje okretanjem cijele ploče. Pri tome ne smijemo zaboraviti čišćenje petlje na kraju same funkcije. Sada možemo vidjeti kako završni proizvod izgleda (Slika 9).

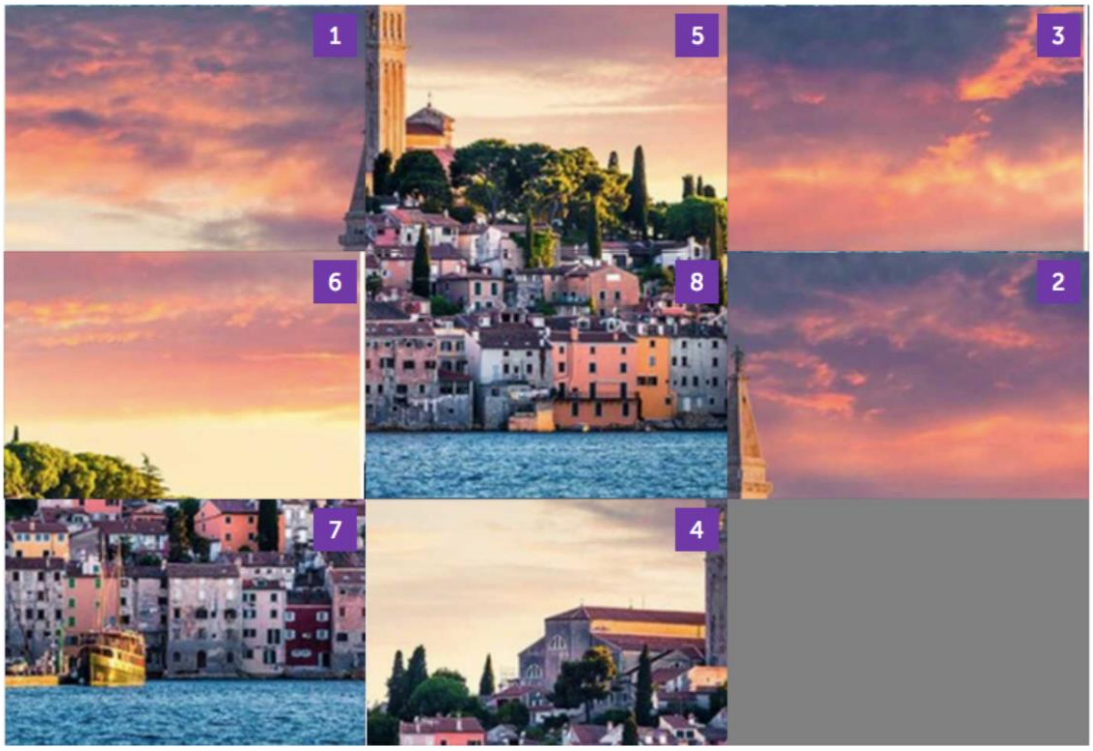


Slika 9 – Igra memorije

Kao što možemo vidjeti na slici 9, kreirana igra memorije sadrži sve što sadrži i igra koju CARNET predlaže za nastavu. Imamo ploču s karticama, gumb za početak, broj poteza te vrijeme koje je proteklo od početka igre. S ovim smo došli do kraja druge igre i dokazali smo da se **JavaScript može upotrijebiti i za izradu igre memorije**.

6. Kreacija klizne slagalice

Treća igra koju ćemo pokušati kreirati uz pomoć JavaScripta je klizna slagalica: igra u kojoj imamo ploču koja se sastoji od 9 dijelova od kojih je jedan dio prazan, a ostali sadržavaju dijelove slike. Cilj igre je spojiti dijelove slike u cijelinu točnim redosljedom tako što kližemo dijelove slagalice jedan po jedan dok ne završimo slagalicu.



Slika 10 – CARNET klizna slagalica

Slika 10 prikazuje kako sama igra izgleda. Samo jedan dio slagalice nije popunjen (sivo polje u donjem desnom kutu) jer uvijek moramo imati jedno slobodno mjesto gdje možemo pomaknuti dijelove slagalice.

Za početak se izrađuje HTML datoteka u kojoj se nalazi kontejner koji sadrži 9 dijelova, po jedan za svaki dio slagalice, a raspoređeni su korištenjem `list item: <li>` oznake unutar HTML koda.

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <meta http-equiv="X-UA-Compatible" content="ie=edge">
8      <link rel="stylesheet" href="css/style.css">
9      <title>Klizna slagalica</title>
10 </head>
11
12 <body>
13     <div id="container">
14         <ul>
15             <li></li>
16             <li></li>
17             <li></li>
18             <li></li>
19             <li></li>
20             <li></li>
21             <li></li>
22             <li></li>
23             <li></li>
24         </ul>
25     </div>
26
27 </body>
28 <script src="js/script.js"></script>
29
30 </html>
```

Kodni blok 32 – HTML kod klizne slagalice

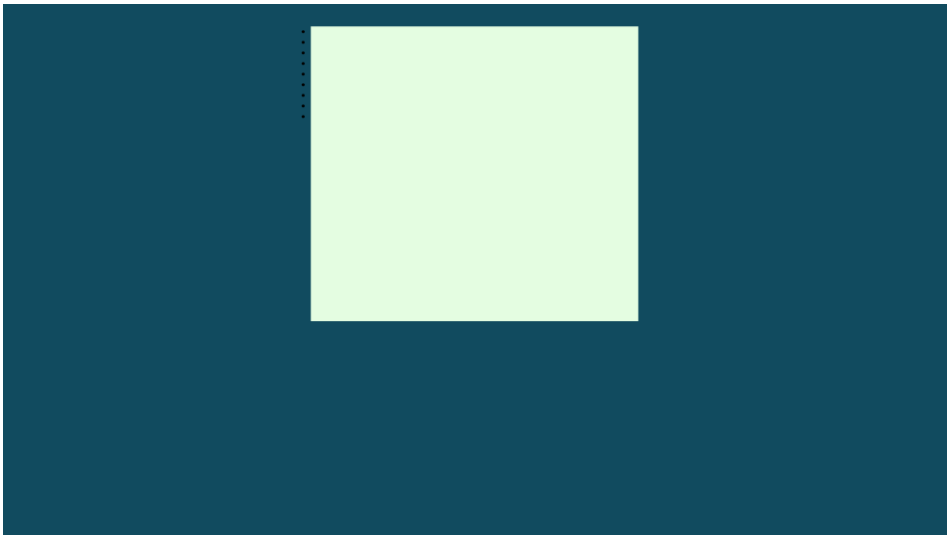
U svaku od ovih oznaka prikazanih na kodnom bloku 32 (kodne linije 15..23) možemo ubaciti jedan dio slagalice dok će jedna ostati prazna. Prije nego što se doda sadržaj unutar tih oznaka potrebno je dodati stil igri koristeći CSS kod.

Za početak se uklanjaju zadane ispune i margine tako što se odabiru svi elemente na stranici koristeći * kao što možemo vidjeti na kodnom bloku 33 (Linija 1).

```
1   * {
2       padding: 0;
3       margin: 0;
4       box-sizing: border-box;
5   }
6   body {
7       display: flex;
8       background-color: #114B5F;
9       text-align: center;
10      flex-direction: column;
11      padding-top: 2%;
12  }
13  #container {
14      width: 500px;
15      height: 500px;
16      background: #E4FDE1;
17      margin: 10px auto;
18  }
```

Kodni blok 33 – Stiliziranje klizne slagalice

Za tijelo igre se dodaje fleksibilan kontejner koji omogućuje lakše stiliziranje stranice (Kodni blok 33 – Linija 6..12). Kontejneru s dijelovima slagalice (Kodni blok 33 – linija 13..18) smo podesili visinu i širinu na 500 piksela te smo mu dodali pozadinsku boju koja pristaje boji pozadine tijela stranice. Dodavanje margine (Kodni blok 33 – linija 17) centrira se kontejner tako što mu dodaje marginu od 10 piksela s vrha i dna te dodaje automatsku odnosno jednaku marginu s lijeve i desne strane. Ukoliko pogledamo kako zasad izgleda igra trebali bismo imati prazni kvadrat na pozadini.



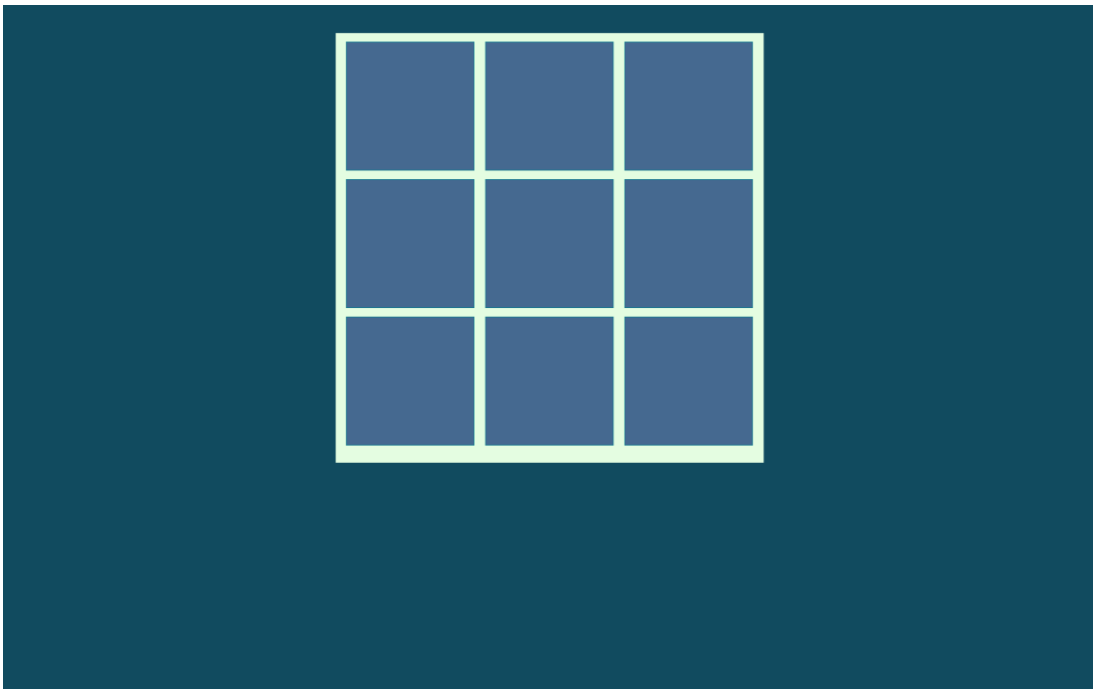
Slika 11 – Prikaz trenutnog stila igre

Sada je potrebno unutar ovoga kvadrata kojeg vidimo na slici 11 ubaciti dijelove slagalice. To ćemo napraviti tako što ćemo stilizirati elemente liste unutar fleksibilnog kontejnera te ćemo ravnomjerno rasporediti same elemente.

```
ul {
    display: flex;
    flex-wrap: wrap;
    list-style-type: none;
    justify-content: space-evenly;
    font-family: 'Audiowide';
    color: #E4FDE1;
}
ul li {
    background: #456990;
    width: 30%;
    height: 150px;
    border: 1px solid #028090;
    margin-top: 10px;
    text-align: center;
    font-size: 2rem;
    padding-top: 3rem;
}
```

Kodni blok 34 – Raspored dijelova slagalice

Kod na kodnom bloku 34 dodaje devet manjih kvadrata koji predstavljaju dijelove slagalice unutar kvadrata koji smo prethodno izradili, te tu promjenu možemo i uočiti ukoliko prikažemo samu igru.



Slika 12 – Prikaz stila dijelova igre

Ono što trebamo znati je da u kliznoj slagalici jedan element uvijek mora biti prazan što za nas trenutno nije slučaj kao što se vidi na slici 12. To se može postići pomoću kreacije klase koja stilizira prazni kvadrat te ćemo unutar HTML koda jednom elementu dodati kod prikazan na kodnom bloku 35.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" href="css/style.css">
    <link href='https://fonts.googleapis.com/css?family=Audiowide' rel='stylesheet'>
    <title>Klizna slagalica</title>
</head>

<body>
    <h1>Klizna slagalica</h1>
    <div id="container">
        <ul>
            <li>1</li>
            <li>2</li>
            <li>3</li>
            <li>4</li>
            <li>5</li>
            <li>6</li>
            <li>7</li>
            <li>8</li>
            <li class="empty"></li>
        </ul>
    </div>
</body>
<script src="js/script.js"></script>

</html>
```

Kodni blok 35 – Dodavanje prazne klase jednom dijelu slagalice

Na kodnom bloku 35 možemo vidjeti da smo za jedan element slagalice dodali klasu kako bi označili da je prazan (Kodni blok 35 – linija 25), a u ostale smo dodali brojeve od 1 do 8 (Kodni blok 35 – linija 17..24). Zatim se prazni dio slagalice stilizira unutar CSS koda dodavajući kod prikazan u kodnom bloku 36.

```
.empty {
    background: #DAD4EF;
    border: 2px solid #114B5F;
}
```

Kodni blok 36 – Stil praznog kvadrata

U kodnom bloku 36 možemo vidjeti CSS kod koji stilizira prazni kvadrat. Također možemo stilizirati i naslov iznad kvadrata kako bi uljepšali slagalicu. Naslov slagalice možemo uočiti na kodnom bloku 35 (Linija 14).

```
47  h1 {
48      color: #E4FDE1;
49      font-family: 'Sofia';
50      text-align: center;
51
52  }
```

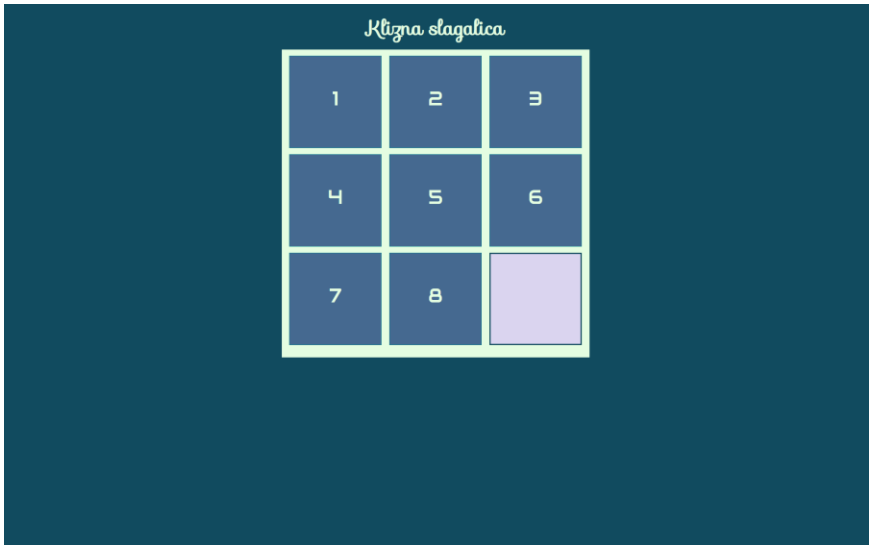
Kodni blok 37 – Dodavanje stila naslovu slagalice

Nakon što smo dodali stil za naslov slagalice kako je prikazano u kodnom bloku 37, dodat ćemo i link za font koji ćemo koristiti za naslov unutar HTML koda.

```
1  <link href='https://fonts.googleapis.com/css?family=Sofia' rel='stylesheet'>
2
3
```

Kodni blok 38 – Dodavanje fonta naslovu igre

S dodavanjem koda prikazanog u kodnom bloku 38 smo završili sa stiliziranjem same igre. Ukoliko pogledamo kako zasad izgleda ova igra, vidjet ćemo da se svaki broj nalazi u svom kvadratu te da se jedan kvadrat razlikuje od ostalih.



Slika 13 – Prikaz završenog stila igre

Sa stiliziranjem igre (prikazanim na slici 13) završenim, možemo započeti s JavaScript dijelom projekta koji će nam omogućiti da klizna slagalica postane funkcionalna.

Kako bi klizna slagalica mogla funkcionirati trebamo omogućiti da se njezini dijelovi mogu povlačiti i ispuštati, a to možemo postići koristeći *drag-and-drop* programsko sučelje. Za povlačenje i ispuštanje dijelova slagalice trebamo definirati rukovatelje događaja[16] unutar JavaScript datoteke.

```
1  const dragstart_handler = ev => {
2      console.log("dragstart")
3      ev.dataTransfer.setData("text/plain", ev.target.id)
4      ev.dataTransfer.dropEffect = "move";
5  }
```

Kodni blok 39 – Početak JavaScript koda klizne slagalice

Rukovatelj `dragstart_handler` koji možemo vidjeti u kodnom bloku 39 se brine za događaj `ev` koji započinje kada korisnik započne s povlačenjem dijela slagalice. Budući da ćemo prenositi podatke tijekom povlačenja, dobit ćemo podatke priložene tom elementu te ih dodati u `dataTransfer` objekt, što je objekt koji koristimo za držanje podataka koje povlačimo tijekom ove operacije.

Rukovatelj `dragover_handler` (Kodni blok 40 – linija 1..4) je jednostavna funkcija, a jedina operacija koju radi je da poziva `preventDefault()` funkciju kako bi spriječili dodatnu obradu ovog događaja. Također odjavljujemo događaj kako bismo mogli pratiti kada je pokrenut. Unutar rukovatelja `drop_handler` (Kodni blok 40 – linija 5..11) dobivamo podatke koje smo pohranili unutar `dataTransfer` objekta te taj podatak prenosimo na element u koji ispuštamo dio slagalice. Uz pomoć `dragend_handler` (Kodni blok 40 – linija 12..16) rukovatelja brišemo podatke unutar `dataTransfer` objekta kada ispustimo element. Sada trebamo promijeniti dio HTML koda kako bismo artiklima unutar liste koji sadrže podatke omogućili pomicanje.

[16] Rukovatelj događaja se može koristiti za rukovanje i provjeru korisničkog unosa, radnji korisnika i radnji preglednika.

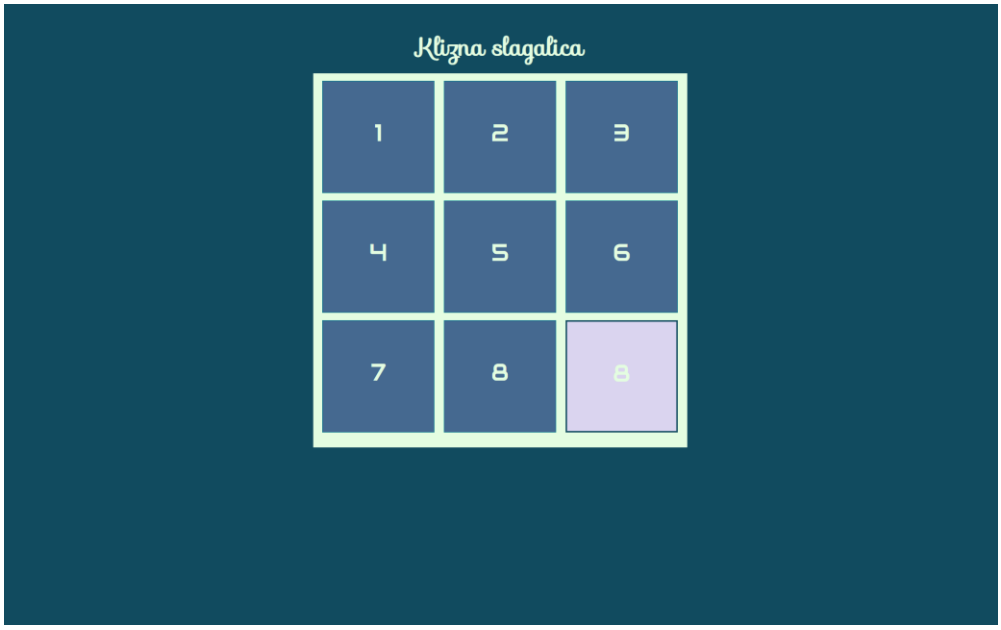
```
const dragover_handler = ev => {
    console.log("dragOver");
    ev.preventDefault();
}
const drop_handler = ev => {
    console.log("drag")
    ev.preventDefault();

    const data = ev.dataTransfer.getData("text/plain");
    ev.target.innerText = document.getElementById(data).innerText;
}
const dragend_handler = ev => {
  console.log("dragEnd");

  ev.dataTransfer.clearData();
}
```

Kodni blok 40 – Rukovatelji događaja za ispuštanje kućice

Stavkama kojima želimo omogućiti pomicanje trebamo dati identifikaciju te im dodati istinitu vrijednost za mogućnost pomicanja kao što vidimo na kodnom bloku 41 te pozvati dijelove koda iz JavaScript datoteke. Također trebamo omogućiti ispuštanje elementa koji držimo na prazno mjesto. Nakon što smo dodali potrebne atribute dijelovima slagalice, možemo ju prikazati da vidimo što smo dobili (Slika 14).

```
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
    <li>6</li>
    <li>7</li>
    <li id="li8" draggable="true" ondragstart="dragstart_handler(event)" ondragend="dragend_handler(event)">8</li>
    <li id="li9" class="empty" ondrop="drop_handler(event);" ondragover="dragover_handler(event);"></li>
</ul>
```

Kodni blok 41 – Omogućavanje pomicanja kućicama



Slika 14 – Prvo pomicanje na ploči

Na slici 14 vidimo da smo uspjeli postići pomicanje i ispuštanje dijela slagalice, ali se susrećemo s novim problemom. Nakon što smo ispustili dio slagalice vidimo da imamo dva jednaka dijela slagalice. To znači da trebamo isprazniti dio slagalice koji smo pomaknuli nakon što ga ispustimo na prazno mjesto. Također ćemo trebati promijeniti stil kućica nakon što se klizanje dijela slagalice dogodi. No prije toga ćemo ažurirati kod kako bi ispraznili kućicu nakon što taj dio slagalice pomaknemo.

```
const drop_handler = ev => {
    console.log("drag")
    ev.preventDefault();

const data = ev.dataTransfer.getData("text/plain");
    ev.target.innerText = document.getElementById(data).innerText;
    document.getElementById(data).innerText = "";
}
```

Kodni blok 42 – Ažuriranje drop_handler rukovatelja

Kod prikazan na kodnom bloku 42 nam omogućuje da se kućice isprazne nakon što su pomaknute koristeći `innerText` parametar, a sada ćemo samu funkcionalnost pomaknuti iz HTML koda u JavaScript datoteku budući da kućice sada sadržavaju sadržaj

koji je definiran unutar HTML-a. Ažurirat ćemo dio HTML koda na način prikazan u kodnom bloku 43.

```
17          <ul>
18              <li></li>
19              <li></li>
20              <li></li>
21              <li></li>
22              <li></li>
23              <li></li>
24              <li></li>
25              <li></li>
26              <li></li>
27          </ul>
```

Kodni blok 43 – Prazna lista unutar HTML koda

Sad kada imamo praznu listu u HTML dijelu koda, unutar JavaScript datoteke ćemo dodati kod prikazan u kodnom bloku 44.

```
1   let ul = document.querySelectorAll('li');;
2   const numbers = ["1", "2", "3", "4", "5", "6", "7", "8", ""]
3
4   const setId = (items) => {
5       for (let i = 0; i < items.length; i++) {
6           items[i].setAttribute("id", `li${i}`)
7       }
8   }
9
10  const fillGrid = (items, numbers) => {
11      items.forEach((item, i) => {
12          item.innerText = numbers[i];
13      })
14  }
15
16  fillGrid(ul, numbers);
```

Kodni blok 44 – Elementi slagalice direktno u JavaScript kodu

Za početak ćemo sve stavke unutar liste ubaciti u popis čvorova koristeći `querySelectAll` funkciju prikazanu na kodnom bloku 44 (Linija 1). Niz `numbers`

sadrži listu slova gdje je zadnja stavka prazan string što ćemo koristiti za praznu kućicu (Kodni blok 44 – linija 2). Funkcija `setID` (Kodni blok 44 – linija 4..8) zadaje identifikacije svim stavkama u listi tako što poziva `setAttribute()` funkciju za svaki od njih, a identifikacija se napravi konkatenacijom prefiksa `list item` elemenata s indeksom u `for` petlji. Unutar `fillGrid` funkcije (Kodni blok 44 – linija 10..14) stvaramo petlju kroz sve `list item` elemente dok pazimo na njihov indeks `i` te ih popunjavamo s vrijednostima navedenima unutar niza `numbers` kojem pristupamo pomoću indeksa `i`. Ukoliko prikažemo ovaj kod on će izgledati isto kao i prije, ali ovoga puta se sve odvija direktno unutar JavaScripta.

Sada su sve kućice ispunjene sadržajem iz niza `numbers` u koje smo upisali brojeve od 1 do 8, ali s trenutnim kodom uvijek ćemo imati jednak početni raspored kućica. Ono što želimo je da se sve kućice izmiješaju tako da uvijek dobivamo nasumičnu konfiguraciju kućica koje moramo posložiti kako bi riješili slagalicu.

Kako bismo izradili funkciju koja će promješati vrijednosti unutar niza treba nam sljedeća logika unutar koda:

- petlja unutar niza,
- za svaki indeks `i` u petlji, odabrati nasumični index `j`,
- zamjeniti elemente u indeksu `i` i `j`.

```
1  const shuffle = (arr) => {
2      const copy = [...arr];
3
4      for (let i = 0; i < copy.length; i++) {
5
6          let j = parseInt(Math.random() * copy.length);
7
8          let temp = copy[i];
9          copy[i] = copy[j];
10         copy[j] = temp;
11     }
12     return copy;
13 }
```

Kodni blok 45 – Funkcija za miješanje kućica

U kodu prikazanom u kodnom bloku 45 (Linija 2) možemo vidjeti da pravimo kopiju niza u funkciji prije nego što izmiješamo stavke koje se nalaze unutar niza (Kodni blok 45 – linija 4..12), a to nam omogućuje da ne izmjenimo sami niz, već da preko funkcije vraćamo potpuno novi niz.

Sada trebamo ažurirati `fillGrid` funkciju da prvo promiješamo niz prije nego podatke ubacimo u kućice.

```
1  const fillGrid = (items, numbers) => {
2      let shuffled = shuffle(numbers);
3
4      items.forEach((item, i) => {
5          item.innerText = shuffled[i];
6      })
7  }
```

Kodni blok 46 – Ažuriranja funkcija za popunjavanje kućica

Kod prikazan na kodnom bloku 46 nam omogućuje da svaki put kada pokrenemo kliznu slagalicu imamo različitu konfiguraciju kućica.

Sada nam je potrebna funkcija koja će uzeti podatke iz liste, pronaći koja je kućica prazna i omogućiti nam da kućice možemo ispustiti na to mjesto. Za početak ćemo izraditi funkciju koja nam omogućuje da sve kućice možemo vući te ćemo ju poslije ažurirati da se povlačenje omogući samo kućicama koje se nalaze pokraj prazne kućice. No prije svega ćemo izraditi `setUp` funkciju iz koje ćemo pozivati sve druge funkcije te ćemo sve dosadašnje funkcije premjestiti unutar ove funkcije.

```
1   let ul = document.querySelectorAll('li');;
2   const numbers= ["1", "2", "3", "4", "5", "6", "7", "8", ""]
3
4   function setUp() {
5       fillGrid(ul, numbers);
6       setId(ul)
7       setDroppable(ul) ;
8       setDraggable(ul);
9
10  }
11
12  const state = {}
13  state.content = numbers;
14
15  const setDroppable = (items) => {
16      items.forEach((item, i) => {
17          if(!item.innerText) {
18              item.setAttribute("ondrop", "drop_handler(event);");
19              item.setAttribute("ondragover", "dragover_handler(event);");
20              item.setAttribute("class", "empty");
21          }
22          return;
23      })
24  }
25
26  const setDraggable = (items) => {
27      items.forEach(item => {
28              item.setAttribute("draggable", "true");
29              item.setAttribute("ondragstart", "dragstart_handler(event)");
30              item.setAttribute("ondragend", "dragend_handler(event)");
31      })
32  }
```

Kodni blok 47 – Funkcija za postavljanje igre

U kod smo također dodali i `state` objekt koji će sadržavati stanje same igre, a vidljiv je u kodnom bloku 47 (Linija 12). Sada trebamo ažurirati HTML kod tako da se pozove `setUp` funkcija kada se tijelo stranice učita.

```
14    <body onload="setUp()">
15        <h1>Klizna slagalica</h1>
```

Kodni blok 48 – Pozivanje funkcije kod učitavanja stranice

Uz pomoć koda prikazanog u kodnom bloku 48, sve funkcije koje se nalaze unutar `setUp` funkcije se pozovu odmah nakon što se stranica učita. Ukoliko pogledamo kako trenutno izgleda klizna slagalica, možemo vidjeti da nam je omogućeno povlačenje kućica na prazno mjesto, ali kada ispustimo kućicu, stil se ne mijenja te i dalje izgleda kao da je ta kućica prazna.

Sada ćemo ažurirati kod tako da kada u praznu kućicu ispustimo sadržaj ona postane popunjena te se na nju više ne može ispuštati. Podsjetimo se da smo koristili `setAttribute` funkciju da zadamo atribut te ćemo sada iskoristiti istu funkciju kako bi uklonili atribut. Kako bi to postigli kao atribut ćemo staviti prazni niz te ćemo poslije toga omogućiti ispuštanje sadržaja u kućicu iz koje smo uklonili atribute.

```
const drop_handler = ev => {
    console.log("drag")
    ev.preventDefault();

    const data = ev.dataTransfer.getData("text/plain");
    ev.target.innerText = document.getElementById(data).innerText;

    ev.target.classList.remove("empty")

    ev.target.setAttribute("ondrop", "");
    ev.target.setAttribute("ondragover", "");
    document.getElementById(data).innerText = "";
}

const dragend_handler = ev => {
  console.log("dragEnd");

  ev.dataTransfer.clearData();

  setDroppable(document.querySelectorAll('li'));
  setDraggable(document.querySelectorAll('li'))
}
```

Kodni blok 49 – Ažuriranje koda za ispuštanje kućica

Ukoliko sada testiramo samu igru vidjet ćemo da svaki put kada povučemo i ispustimo kućicu, izvorna kućica se isprazni, a destinacija se popuni, a to smo postigli uz

pomoć koda prikazanog na kodnom bloku 49. No još uvijek se mogu povlačiti i ispuštati kućice koje nisu pokraj prazne kućice. Taj dio ćemo ispraviti poslije a za sada ćemo se fokusirati na stanje i dimenziju igre.

Svaki puta kada povučemo i ispustimo sadržaj kućice, samo uređenje kućica se mijenja, a sve promjene želimo pratiti unutar `state` objekta. Izradit ćemo funkciju koja dobiva to stanje te ćemo napraviti funkciju koja će pohranjivati vizualni stil kućica kao niz nizova. Ažurirat ćemo JavaScript kod na način prikazan u kodnom bloku 50.

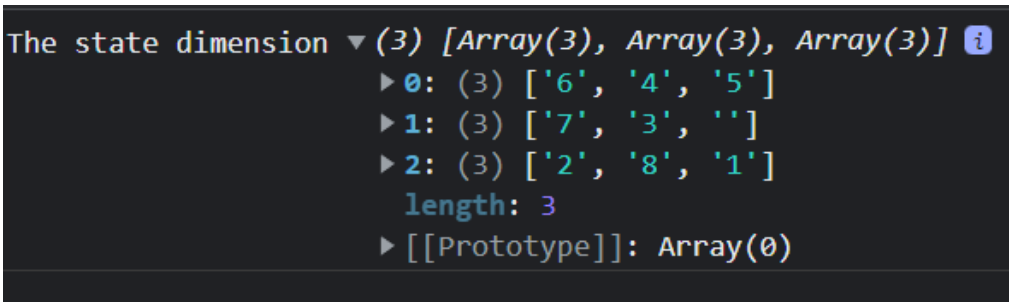
Funkcija `getDimension` prikazana na kodnom bloku 50 (Linija 22..31) prolazi kroz sadržaj (niz od devet znakova) tri puta, svaki put izreže tri dijela sadržaja i dodaje ih u niz, a to nam stvar niz od tri niza. Prvi niz predstavlja prvi red igre, drugi niz drugi red i treći niz predstavlja treći red same igre. Ovo znači da nam se trenutno stanje igre pohranjuje u dva formata gdje `state.content` pohranjuje kućice u linearnom slijedu odnosno kao jednodimenzionalni niz dok `state.dimension` pohranjuje kućice u dvodimenzionalnom nizu što je zapravo vizualni prikaz same igre.

```
function setUp() {
    fillGrid(ul, numbers);
    setId(ul)

    state.content = getState(ul);
    state.dimension = getDimension(state);

    setDroppable(ul) ;
    setDraggable(ul);
    console.log("The state dimension", state.dimension)
}
const state = {}
state.content = numbers;

const getState = (items) => {
    const content = [];
    items.forEach((item, i) => {
        content.push(item.innerText)
    });
    return content;
}
const getDimension = (state) => {
    let j = 0;
    let arr = [];
    const {content} = state;
    for(let i = 0; i < 3; i++) {
        arr.push(content.slice(j, j+3));
        j+=3;
    }
    return arr;
}
const setDroppable = (items) => {
    items.forEach((item, i) => {
        if(!item.innerText) {
            state.emptyCellIndex = i;
            item.setAttribute("ondrop", "drop_handler(event);");
            item.setAttribute("ondragover", "dragover_handler(event);");
            item.setAttribute("class", "empty");
        }
        return;
    })
}
```

Kodni blok 50 – Stanje i dimenzija igre

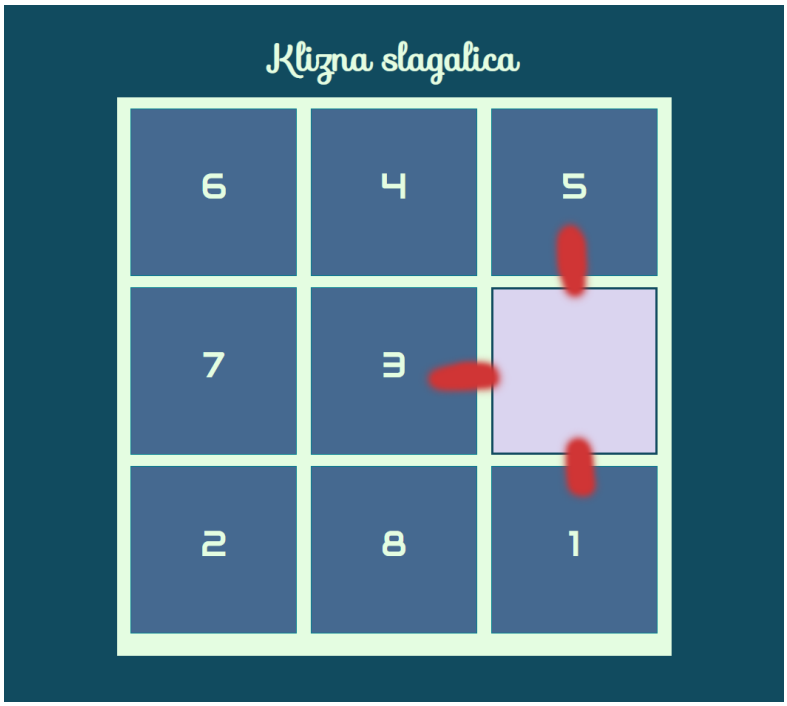
Unutar `setUp` funkcije odjavljujemo izlaz `getDimension` funkcije te na slici 15 vidimo da se vizualni prikaz pohranjuje u dvodimenzionalnom nizu. Također smo dodali novu liniju unutar `setDroppable` funkcije kako bi pohranili indeks prazne kućice u

stanje igre pomoću funkcije `state.emptyCellIndex = i`. Indeks koji ovdje pohranjujemo je indeks iz jednodimenzionalnog niza same stavke unutar liste.



Slika 15 – Prikaz dimenzije u konzoli

Sada trebamo doći do prazne kućice kako bi samo susjednim kućicama, kao što su prikazane na slici 16, mogli omogućiti povlačenje. Trenutno možemo povlačiti bilo koju kućicu na prazno mjesto. Kako bi to promijenili kreirat ćemo `getEmptyCell` funkciju koja će vraćati dvodimenzionalnu poziciju iz *`state.dimension`* niza. Indeks prazne kućice već imamo pohranjen unutar `state.emptyCellIndex` gdje se nalazi jednodimenzionalni indeks prazne kućice.



Slika 16 – Susjedne kućice

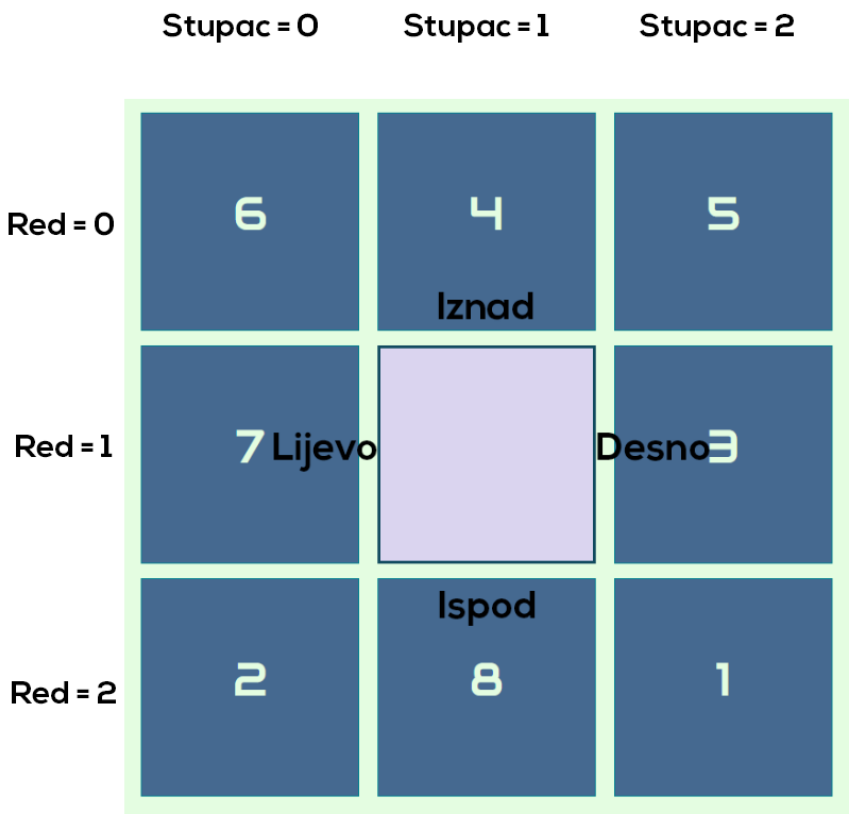
S obzirom da nam je potrebna dvodimenzionalna pozicija trebamo pomoć matematike kako bi izračunali tu poziciju prazne kućice od jednodimenzionalnog indeksa. Dvodimenzionalna pozicija igre sadrži tri reda i tri stupca što obuhvaća svih devet kućica. `state.emtyCellIndex` sadrži linearni indeks prazne kućice, a budući da indeksi niza uvijek počinju od nule, točna pozicija prazne kućice u nizu će biti `state.emptyCellIndex+1`.

Kako bi dobili red u kojem se kućica nalazi podijelit ćemo broj kućice sa 3 što je broj kučica u redu. Kao rezultat dobivamo broj u redu u koji pada prazna kućica. Pronalaženje u kojem stupcu se prazna kućica nalazi je malo teže, ali uz pomoć matematičkih radnji možemo doći do rezultata. Pomnožit ćemo broj pohranjen unutar `emptyCellRow` sa 3 te zatim od rezultata oduzeti `emptyCellNumber` te ćemo zatim taj rezultat oduzeti od broja 3. Kreirat ćemo `getEmptyCell` funkciju odmah nakon `getState` funkcije.

```
1  const getEmptyCell = () => {
2      const emptyCellNumber = state.emptyCellIndex+1;
3      const emptyCellRow = Math.ceil(emptyCellNumber/3);
4      const emptyCellCol = 3 - (3 * emptyCellRow - emptyCellNumber);
5      return [emptyCellRow-1, emptyCellCol-1]
6  }
```

Kodni blok 51 – Funkcija za poziciju prazne kućice

Budući da sada imamo način na koji dolazimo do pozicije prazne kućice, a prikazan je na kodnom bloku 51, imamo i način na koji možemo omogućiti povlačenje samo kućicama koje se nalaze pokraj prazne kućice. Budući da prazna kućica može imati kućice iznad, ispod te s lijeve ili desne strane, samo tim kućicama bi trebali omogućiti povlačenje. Kako bi izračunali indekse kućica susjednih praznoj prvo možemo pogledati primjer prikazan na 16.



Slika 17 – Primjer prazne kućice

Na slici 17 vidimo da se prazna kućica nalazi u sredini, to jest u drugom redu i drugom stupcu, ali budući da u funkcijama koje koristimo uvijek oduzimamo 1 iz razloga što indeksi niza uvijek počinju od nula, možemo reći da se nalazi u prvom redu i prvom stupcu. Susjedne kućice se nalaze sa sve četiri strane što bi značilo da se kućica s lijeva i desna nalaze u prvom redu, a kućica s lijeva je u nultom stupcu dok je kućica s desna u drugom stupcu. Tako se i kućice iznad i ispod nalaze unutar prvog stupca, kućica iznad u nultom redu, a kućica ispod u drugom redu. Sada ćemo ažurirati `setDraggable` funkciju na način prikazan na kodnom bloku 52.

```
1  const setDraggable = (items) => {
2      const [row, col] = getEmptyCell();
3
4      let left, right, top, bottom = null;
5      if(state.dimension[row][col-1]) left = state.dimension[row][col-1];
6      if(state.dimension[row][col+1]) right = state.dimension[row][col+1];
7      if(state.dimension[row-1] != undefined) top = state.dimension[row-1][col];
8      if(state.dimension[row+1] != undefined) bottom = state.dimension[row+1][col];
9
10     items.forEach(item => {
11         if(item.innerText == top ||
12             item.innerText == bottom ||
13             item.innerText == right ||
14             item.innerText == left) {
15                 item.setAttribute("draggable", "true");
16                 item.setAttribute("ondragstart", "dragstart_handler(event)");
17                 item.setAttribute("ondragend", "dragend_handler(event)")
18             }
19
20     })
21 }
```

Kodni blok 52 – Omogućenje povlačenja susjednim kućicama

Sada pomoću `setDraggable` funkcije prikazane na kodnom bloku 52 dolazimo do susjednih kućica te samo njima omogućavamo povlačenje (Kodni blok 52 – Linija 10..20). Ukoliko testiramo samu igru možemo vidjeti da se samo kućice susjedne onoj praznoj mogu povlačiti. No, ukoliko pogledamo malo detaljnije, možemo vidjeti da se i nakon povlačenja i dalje mogu povlačiti samo kućice koje su bile susjedne prvoj praznoj kućici. Kako bi to popravili, trebamo ažurirati kod tako da stanje i dimenziju igre ažuriramo svaki puta kada završimo sa povlačenjem i ispuštanjem jedne kućice.

```
1   const drop_handler = ev => {
2       console.log("drag")
3       ev.preventDefault();
4       const data = ev.dataTransfer.getData("text/plain");
5       ev.target.innerText = document.getElementById(data).innerText;
6
7       ev.target.classList.remove("empty")
8       ev.target.setAttribute("ondrop", "");
9       ev.target.setAttribute("ondragover", "");
10      document.getElementById(data).innerText = "";
11
12      state.content = getState(ul);
13      state.dimension = getDimension(state);
14  }
```

Kodni blok 53 – Ažuriranje koda za povlačenje

Problem koji smo imali smo rješili uz pomoć koda prikazanog u kodnom bloku 53, no i dalje postoji problem koji je dosta sličan ovome, a to je da sve kućice koje u jednom trenutku imaju omogućeno povlačenje, tu mogućnost imaju zauvijek. Kako bi se riješili tog problema kreirat ćemo funkciju `removeDroppable` koja će onemogućiti povlačenje kućici kada je popunjena. Tu funkciju ćemo pozvati unutar `dragend_handler` funkcije te ćemo ju kreirati nakon `setDroppable` funkcije (kodni blok 54).

Uz kod prikazan u kodnom bloku 54 koji onemogućuje povlačenje kućicama korištenjem `removeDroppable` funkcije (Kodni blok 54 – linija 1..9), igra je skoro gotova. Preostaje još samo provjera je li korisnik točno poredao kućice. Ipak ćemo prije toga morati rješiti još jedan mali problem. Taj problem je da klizna slagalica sadrži jedno svojstvo na koje ćemo morati utjecati, a to je da se ona sastoji od osam dijelova koje trebamo poredati točnim redosljedom, ali nije svaka konfiguracija kućica rješiva. Budući da se kućice nasumično poredaju na početku igre, moguće je da se stvori konfiguracija koja nije rješiva.

```
const removeDroppable = (items) => {
    items.forEach((item) => {
        item.setAttribute("ondrop", "");
        item.setAttribute("ondragover", "");
        item.setAttribute("draggable", "false");
        item.setAttribute("ondragstart", "");
        item.setAttribute("ondragend", "");
    })
}

....

const dragend_handler = ev => {
  console.log("dragEnd");
  ev.dataTransfer.clearData();
  removeDroppable(document.querySelectorAll('li'));
  setDroppable(document.querySelectorAll('li'));
  setDraggable(document.querySelectorAll('li'))
}
```

Kodni blok 54 – Funkcija za onemogućavanje povlačenja

Kako uopće možemo reći je li konfiguracija rješiva ili ne? Zagonetka koja se sastoji od 8 dijelova je rješiva samo ukoliko je broj inverzija paran broj u konfiguraciji. Kako bi shvatili što to znači prvo moramo definirati što nam znači inverzija. Inverzija se pojavljuje kada je par kućica u obrnutom rasporedu od onog u točnoj konfiguraciji. To ćemo lakše shvatiti uz pomoć primjera prikazanog u kodnom bloku 55.

```
[
    ["1", "2", "3"],
    ["4", "", "5"],
    ["7", "6",  "8"]
]
```

Kodni blok 55 – Inverzija

U primjeru prikazanom na kodnom bloku 55 se javljaju dvije inverzije, a to su inverzija (7, 6) i inverzija (7, 8), a budući da je broj inverzija dva, što je paran broj, ova konfiguracija će biti rješiva. Sada trebamo kreirati funkciju koja će nalaziti broj inverzija u konfiguraciji. To ćemo postići tako da ćemo izabrati svaki element i usporediti ga s elementima koji dolaze iza, a ako je prvi element „veći“ od drugog javila se inverzija. Kreirat ćemo `isSolvable` funkciju ispred `fillGrid` funkcije te ćemo ažurirati `fillGrid` funkciju da ispuni kućice samo u slučaju ako je konfiguracija rješiva.

```
const isSolvable = (arr) => {
    let number_of_inv = 0;
    for(let i =0; i<arr.length; i++){
        for(let j = i+1; j < arr.length; j++) {
            if((arr[i] && arr[j]) && arr[i] > arr[j]) number_of_inv++;
        }
    }
    return (number_of_inv % 2 == 0);
}

const fillGrid = (items, letters) => {
    let shuffled = shuffle(letters);
    while(!isSolvable(shuffled)) {
        shuffled = shuffle(letters);
    }

    items.forEach((item, i) => {
        item.innerText = shuffled[i];
    })
}
```

Kodni blok 56 – Funkcija za rješive konfiguracije

Uz pomoć koda prikazanog u kodnom bloku 56 smo sigurni da će se svaka konfiguracija moći rješiti te ćemo sada provjeriti kada je korisnik rješio slagalicu. Kreirat ćemo `isCorrect` funkciju koja će uzimati podatke iz `state.content` i `numbers`, konvertirati te podatke u nizove znakova, zatim ih usporediti i provjeriti jesu li ti podaci jednaki. Funkciju kreiramo nakon `isSolvable` funkcije.

Budući da se provjera rješenja slagalice, odnosno funkcija prikazana u kodnom bloku 57, događa nakon što ispustimo kućicu trebamo ažurirati `dragend_handler` funkciju.

```
1  const isCorrect = (solution, content) => {
2      if(JSON.stringify(solution) == JSON.stringify(content)) return true;
3      return false;
4  }
```

Kodni blok 57 – Funkcija točnog rješenja

Ukoliko pokrenemo igricu i točno rješimo slagalicu, poruka da je došlo do rješenja se pojavljuje u konzoli kao što možemo vidjeti u kodnom bloku 58 (Linija 9). No, mi ne želimo da igrači moraju provjeravati konzolu da bi saznali da su završili sa slagalicom. Stoga ćemo kreirati poruku koja će iskočiti na ekranu kada je slagalica rješena. Kako bi do toga došli, trebamo ažurirati HTML kod, CSS kod i JavaScript kod.

```
1   const dragend_handler = ev => {
2       console.log("dragEnd");
3       ev.dataTransfer.clearData();
4       removeDroppable(document.querySelectorAll('li'));
5       setDroppable(document.querySelectorAll('li'));
6       setDraggable(document.querySelectorAll('li'))
7
8       if(isCorrect(letters, state.content)) {
9           console.log("Pobjeda!");
10      }
11  }
```

Kodni blok 58 – Ažuriranje funkcije za ispuštanje kućice

HTML kod ćemo urediti kako je prikazano u kodnom bloku 59.

```
1  <body onload="setUp()">
2      <h1>Klizna slagalica</h1>
3      <div id="modal" class="hide">
4          <div id="header"><button id="closeBtn" onclick="hideModal()">x</button></div>
5          <h1 id="message">Pobjeda!</h1>
6      </div>
7      <div id="container">
```

Kodni blok 59 – Ažurirani HTML kod

CSS kod ćemo ažurirati kako je prikazano u kodnom bloku 60.

```css
#modal {
    width: 40%;
    height: 300px;
    background: #5af0ee;
    position: absolute;
    top: 30%;
    left: 50%;
    transform: translateX(-50%);
    font-family: 'Sofia';
}

#modal #message {
    color: #456990;
    margin: 0;
    font-size: 4rem;
}

.hide {
    display: none;
}

#modal #header {
    text-align: right;
    padding: 5px 10px;
}

#modal #header button {
    padding: 10px;
    border: 0px;
    background: #456990;
    color: #E4FDE1;
    cursor: pointer;
}
```

Kodni blok 60 – Ažurirani CSS kod

JavaScript ćemo ažurirati kako je prikazano u kodnom bloku 61.

```
const dragend_handler = ev => {
    console.log("dragEnd");
    ev.dataTransfer.clearData();
    removeDroppable(document.querySelectorAll('li'));

    setDroppable(document.querySelectorAll('li'));
    setDraggable(document.querySelectorAll('li'))

      if(isCorrect(letters, state.content)) {
          showModal();
      }
  }

  const showModal = () => {
      document.getElementById('message').innerText = "Pobjeda!";
      document.getElementById('modal').classList.remove("hide");

  }

  const hideModal = () => {
      document.getElementById('modal').classList.add("hide");
  }
```

Kodni blok 61 – Ažurirani JavaScript kod

Ovaj dio koda je veoma jednostavan, imamo dio HTML koda (Kodni blok 59) kojem smo dali identifikator te mu omogućili sakrivanje nakon što se na njega klikne. Unutar CSS koda (Kodni blok 60) smo mu dodali stil kako bi se razlikovao od pozadine te taj dio, kreiran unutar HTML koda, pozivamo unutar funkcije u JavaScript dijelu (Kodni blok 61) ukoliko je došlo do točnog rješenja.

To bi bio kraj kreacije klizne slagalice. Ukoliko ju pokrenemo i završimo možemo vidjeti kako izgleda (Slika 18).

Slika 18 – Završetak klizne slagalice

Ukoliko izrađenu kliznu slagalicu (Slika 18) usporedimo s CARNET-ovom verzijom vidimo da imamo sve potrebne dijelove, 8 pomičnih dijelova i deveti dio na koji možemo pomaknuti susjedne kućice. U slučaju rješavanja dobivamo poruku da smo uspješno riješili slagalicu. S ovim smo dokazali da se **JavaScript može koristiti za kreaciju i ovog tipa igre**.

7. Zaključak

Kao što smo imali prilike vidjeti u detaljnim opisima tijeka izrade ovih igara (kviza znanja, igre memorije i klizne slagalice), JavaScript je dovoljno razvijen u ovom trenutku da se s njim mogu kreirati edukativne igre. Kako vrijeme prolazi, JavaScript se koristi za sve više i više stvari te se u pozadini gotovo svega na internetu pojavljuje u nekoj mjeri, pa tako i u pozadini mrežnih igara.

Naravno nijedan programski jezik nije savršen, ali kako bi se pobrinuli za njegove nedostatke, kao pomoć koristimo HTML i CSS programske jezike koji nam omogućuju jednostavniji način za prikaz i stiliziranje samog koda. Koristeći kombinaciju ta tri jezika možemo kreirati gotovo sve što možemo naći na internetu, ali za to je potrebno opsežno znanje svih triju jezika.

Što se tiče edukativnog aspekta mrežnih igara one pomažu i kod razvoja vizualizacije, što je Crown istraživao 2013 godine na studentima grafičkog inženjeringa. Za potrebe ovog rada smo izabrali tri vrste igara koje su veoma korisne učenicima. Tako se **kviz znanja** najčešće koristi iz razloga što je jednostavan za izradu i zabavan za korištenje, a uz njegovu pomoć možemo i jednostavno pratiti rezultate budući da na kraju rješavanja dobivamo povratnu informaciju o točnim, odnosno, netočnim odgovorima. **Igra memorije** pomaže učenicima da razvijaju sposobnost pamćenja koja im uvelike pomaže kod učenja, a ta vrsta igre je veoma zanimljiva i potiče ih na rad te dovodi do pozitivnih natjecanja među učenicima budući da svatko želi postići što bolji rezultat. **Klizna slagalica** je korisna kada znanje želimo prenijeti i uz pomoć vizualnog prikaza, a ovakav tip igre može biti koristan upravo kod predmeta gdje je to potrebno, a također i nakon što učenici rješe slagalicu, možemo dodati i informaciju o slici koju su sastavili tako da će i oni smatrati da su zaslužili to znanje.

S tim smo došli do kraja ovog rada sa zaključcima da se JavaScript može koristiti u kreaciji edukativnih igara te nema nepremostivih problema s različitim elementima koji

su potrebni za izradu opisanih igara. Zbog svojih pozitivnih utjecaja na učenje, kao i jednostavnosti izrade opisanim programskim rješenjima (HTML, CSS i JavaScript) možemo očekivati da ćemo viđati sve više edukativnih igara u budućnosti obrazovnog procesa.

Literatura

1. Allen, L. K., Crossley, S. A., Snow, E. L., McNamara, D. S. (2014). L2 writing practice: Game enjoyment as a key to engagement. U:Language Learning & Technology 18.2, str. 124-150.
2. Al-Mashaqbeh, I., Al Dweri, A. (2014). Educational math game software: A supporting tool for first grade students' achievement. U: Journal of Education and Practice 5.5, str. 134-141.
3. Aljezawi, M., Albashtawy, M. (2015). Quiz game teaching format versus didactic lectures. U:British Journal of Nursing 24.2, str. 86-92.
4. Annetta, L. (2008). Serious educational games: From theory to practice. U:BRILL
5. Antonova, A., Bontchev, B. (2019). Exploring puzzle-based learning for building effective and motivational maze video games for education. U:Proceedings of 11th Annual International Conference on Education and New Learning Technologies, str. 2425-2434.
6. Brackeen, D., Barker, B., Vanhelsuwé, L. (2004). Developing games in Java. U:New Riders.
7. Cheng, M. T., Huang, W. Y., Hsu, M. E. (2020). Does emotion matter? An investigation into the relationship between emotions and science learning outcomes in a game-based learning environment. U:British Journal of Educational Technology 51.6, str. 2233-2251.
8. Crown, S. W. (2001). Improving visualization skills of engineering graphics students using simple JavaScript web based games. U: Journal of Engineering Education 90.3, str. 347-355.
9. De Freitas, S. (2018). Are games effective learning tools? A review of educational games. U:Journal of Educational Technology & Society 21, str. 74-84.
10. Egges, A. (2014). Building JavaScript Games: For Phones, Tablets, and Desktop. U:Apress 29.

11. Fisher, R. A., Yates, F. (1953). Statistical tables for biological, agricultural and medical research. U:Hafner Publishing Company.
12. Gordon, A. K. (1970). Games For Growth; Educational Games in the Classroom. U:Science Research Associates, College Division.
13. Habgood, M. J., Ainsworth, S. E. (2011). Motivating children to learn effectively: Exploring the value of intrinsic integration in educational games.“ U:The Journal of the Learning Sciences 20.2,str. 169-206.
14. Hajdarević, E., Vučković, K., Dovedan, Z. (2006). Računalo ili raćunalo uz pomoć računala. U: Proceedings of the 29th International Convention MIPRO, Rijeka, str. 283-287
15. Hawkes, R. (2011). Foundation HTML5 Canvas: For Games and Entertainment. U:Apress.
16. Heeter, C., Winn, B., Winn, J., Bozoki, A. (2008). The challenge of challenge: Avoiding and embracing difficulty in a memory game. U:Meaningful Play Conference, East Lansing, Michigan.
17. Mester, G., Molcer, P. S., Delic, V. (2011). Educational games. U:Computer Games as Educational and Management Tools: Uses and Approaches. IGI Global, str. 247-262.
18. Najdi, S., El Sheikh, R. (2012). Educational games: do they make a difference?. U:Procedia-Social and Behavioral Sciences 47, str. 48-51.
19. Noemí, P. M., Máximo, S. H. (2014). Educational games for learning. U:Universal Journal of Educational Research 2.3, str. 230-238.
20. Sivakumar, R. (2022). Effectiveness Of Memory Game On Academic Performance Of Primary School Students. U:Pavlo Tychyna Uman State Pedagogical University, 15.
21. Wang, T. H. (200(). Web-based quiz-game-like formative assessment: Development and evaluation. U:Computers & Education 51.3, str. 1247-1263.
22. Wilton, P. (2004). Beginning JavaScript. U:John Wiley & Sons.

23. Zeng, J., Parks, S., Shang, J. (2020). To learn scientifically, effectively, and enjoyably: A review of educational games. U:Human Behavior and Emerging Technologies 2.2, str. 186-195.
24. Zirawaga, V. S., Olusanya, A. I., Maduku, T. (2017). Gaming in education: Using games as a support tool to teach history. U:Journal of Education and Practice 8.15, str. 55-64.
25. Zwick, U., Paterson, M. S. (1993). The memory game. U:Theoretical computer science 110.1, str. 169-196.

Internetski izvori:

1. CARNETova stranica s prijedlozima igara: https://e-laboratorij.carnet.hr/category/igre/
2. Match the Memory: https://matchthememory.com
3. Najpopularniji programi za izradu igara: https://theninehertz.com/blog/game-engines
4. Najpopularniji programski jezici: https://www.devjobsscanner.com/blog/top-8-most-demanded-languages-in-2022/
5. Puzzel.org: https://puzzel.org
6. Wordwall: https://wordwall.net/myactivities
7. Unity: https://unity.com
8. Unreal Engine: https://www.unrealengine.com

Sažetak

JavaScript je najpopularniji programski jezik u svijetu te je poznat po tome da se koristi u svrhu izrade mrežnih stranica. Sam JavaScript se koristi za puno više od toga te će se ovaj rad fokusirati na jednu zanimljivu stranu tog jezika, a to je uporaba JavaScripta za izradu igara u domeni edukacije. Edukativne igre postaju sve popularnije i sve više se koriste u nastavi, a CARNET predlaže odličan izbor edukativnih igara izrađenih s ciljem korištenja u nastavi za poboljšanje nastavnog procesa. Ovaj rad će na primjeru CARNETovih predloženih igara pokazati da li se JavaScript može koristiti za izradu svih elemenata edukativnih igara ili se ipak može koristiti samo za neke dijelove samih igara. Rad se fokusira na izradi tri vrste igara: kviz znanja, igra memorije i klizna slagalica.

Ključne riječi: *JavaScript, edukativne igre, kviz znanja, igra memorije, klizna slagalica, mrežne igre, programski jezik*

JavaScript in the context of the domain of educational games

Abstract

JavaScript is the most popular programming language in the world and is known for its usage in the making of web pages. JavaScript, however, is being used for a lot of different things and this paper is goin to focus on one of the interesting sides of the language, and that is the usage of JavaScript in the creation of games in the domain of education. Educational games are becoming increasingly popular and sees more usage inside of the classrom and CARNET suggests a great selection of educational games created with the purpose of improving the teaching process. This paper will, on the example of games suggested by CARNET, show if JavaScript can be used to create all of the elements of the educational games, or just some of the parts of those games. The paper will focus on creating three different games suggested, and those are: knowledge quiz, memory game and a sliding puzzle.

Key words: *JavaScript, educational games, knowledge quiz, memory game, sliding puzzle, web games, programming language*