# UPORABA NEKONVENCIONALNIH UPORABNIŠKIH VMESNIKOV NA OSNOVI RAČUNALNIŠKEGA VIDA NA PODROČJU RAČUNALNIŠKIH IGER

**Rok Kreslin, Dejan Dežman, Žiga Emeršič, Peter Peer**

**Laboratorij za računalniški vid,**
**Fakulteta za računalništvo in informatiko, Univerza v Ljubljani,**
**Tržaška 25, 1000 Ljubljana**

**POVZETEK**: *V prispevku so skozi igre 3D labirint, Mehurčki in Navidezni slikar prikazane možnosti uporabe računalniškega vida na področju računalniških iger. Vsem predstavljenim igram je skupno to, da interakcija med igralcem in igro ne poteka preko tipkovnice ali miške, ampak preko spletne kamere. Igre tako krmilimo s premiki telesa.*

## 1. UVOD

Področje računalniških iger predstavlja pomembno vejo računalniške industrije. Uporabnike iger najdemo tako med otroci kot med odraslimi. Razvoj iger je v zadnjih letih temeljil predvsem na izboljšavi grafičnih in zvočnih elementov. Proizvajalci iger želijo igralcu vliti čimbolj realen občutek igranja. Ena izmed osnovnih komponent računalniških iger se praktično ni spremenila od začetkov računalniških iger do danes. Gre za interakcijo med igralcem in računalniško igro. Skorajda samoumevno je, da to interakcijo predstavljata tipkovnica in miška, oziroma v nekaterih primerih igralna palica ali volan.

Premike na tem področju najdemo pri igralnih konzolah. Konzola Wii uporablja tehnologijo Wii Remote [1], ki omogoča krmiljenje iger s pomočjo premikanja majhne naprave v roki uporabnika; položaj naprave se ugotavlja s pomočjo infrardečih žarkov. Applov iPhone pri nekaterih igrah izkorišča vgrajen merilec pospeškov [2]. Pristop, ki temelji na uporabi kamere, so uporabili pri konzoli Playstation 2 [3]. Takšen pristop je zanimiv, saj odpravlja potrebo po dodatni opremi v rokah igralca. Uporabnik upravlja igro s pomočjo premikov telesa, ki jih posname kamera. Zatem procesna enota analizira sliko iz kamere in izvede potrebne premike v igri.

V tem članku so predstavljene tri preproste igre, ki jih krmilimo s pomočjo telesa in spletne kamere. Na konkretnih igrah smo želeli preveriti, kako težavna je implemenentacija takšnih uporabniških vmesnikov in kako dobro se ti obnesejo v praksi. V razdelku 2 je predstavljena igra 3D labirint, v razdelku 3 igra Mehurčki in v razdelku 4 igra Navidezni slikar. Na koncu članka je podan zaključek z oceno uporabniške izkušnje in evalvacijo uporabljenega pristopa.

# 2. 3D LABIRINT

## 2.1 Opis

Labirint je konstrukt, zgrajen iz skupka zapletenih hodnikov ali drugih ovir. Cilj subjekta v labirintu je kar najhitreje najti pot iz njega. V računalniških igrah se za premikanje po labirintu največkrat uporablja tipkovnica ali igralna palica. Tak način igranja igralcu ne omogoča, da bi se v igro zares vživel. Naša rešitev skuša uporabniku približati občutek realnega labirinta. Za premikanje po labirintu igralec uporablja preproste gibe telesa.

Platformo sestavljata aktivni in pasivni modul. Aktivni modul skrbi za procesiranje in prikazovanje slike, medtem ko pasivni modul omogoča enostavno konstruiranje virtualnega labirinta. V nadaljevanju članka se osredotočimo predvsem na aktivni modul, saj je ta bolj zanimiv s stališča računalniškega vida. Krajši opis pasivnega modula je podan zgolj površinsko v razdelku 2.2.2.

## 2.2 Implementacija

### *2.2.1 Aktivni modul*

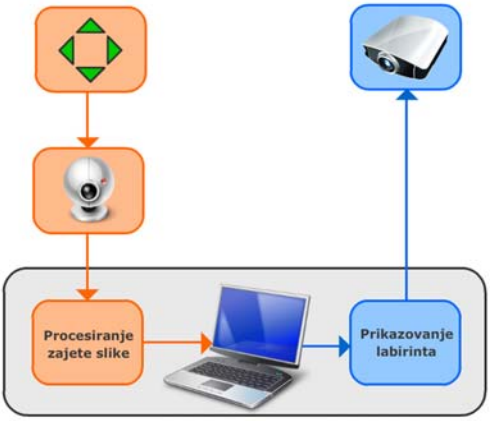
Zgradbo aktivnega modula prikazuje shema na sliki 1.

Sestavljajo ga naslednje komponente:

- igralna plošča,
- enota za zajem slike,
- enota za procesiranje slike,
- enota za prikazovanje labirinta,
- izhodna naprava.

Igralna plošča deluje kot uporabniški vmesnik z gumbi naprej, nazaj, levo, desno. Uporabnik z ustreznimi premiki po igralni plošči povzroči premik v labirintu. Za zajem slike uporabljamo standardno spletno kamero z ločljivostjo 320×240 slikovnih elementov.

Programska rešitev je sestavljena iz enote za procesiranje vhodne slike in enote za prikazovanje labirinta. Prva je implementirana v programskem jeziku Java in za zajem slike uporablja knjižnico JMF [4]. Enota za prikazovanje labirinta je implementirana v programskem okolju Flash. Za delo s 3D objekti uporabljamo knjižnico Sandy 3D [5]. Podrobnosti implementacije enote za prikazovanje v tem prispevku niso predstavljene.

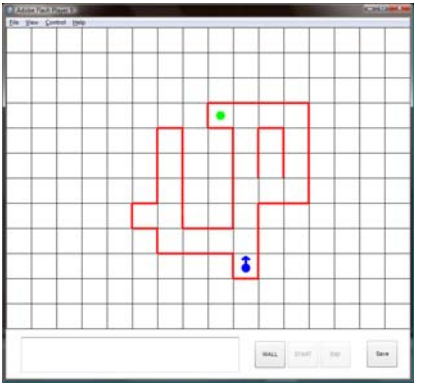



Slika 1: Zgradba aktivnega modula.

Programski enoti sta povezani s pomočjo vtičnic [6,7]. Enota za procesiranje na podlagi vhodne slike ugotovi, ali uporabnik stoji na kateri od komponent igralne plošče. Ta podatek preko vtičnice posreduje enoti za prikazovanje, ki reagira z ustrezno spremembo na izhodni napravi.

### *2.2.2 Pasivni modul*

Pasivni modul je implementiran v programskem okolju Flash. Omogoča hitro in učinkovito konstruiranje virtualnega labirinta. Uporabnik v grafičnem uporabniškem vmesniku vstavlja zidove v koordinatni sistem, nakar modul zapiše koordinate zidov v datoteko, ki jo uporablja prikazovalna enota aktivnega modula. Slika 2 prikazuje zaslonski posnetek uporabniškega vmesnika.



Slika 2: Konstruiranje virtualnega labirinta; rdeče daljice označujejo zidove, modra pika označuje začetno pozicijo, zelena pika označuje cilj.

### *2.2.3 Algoritem*

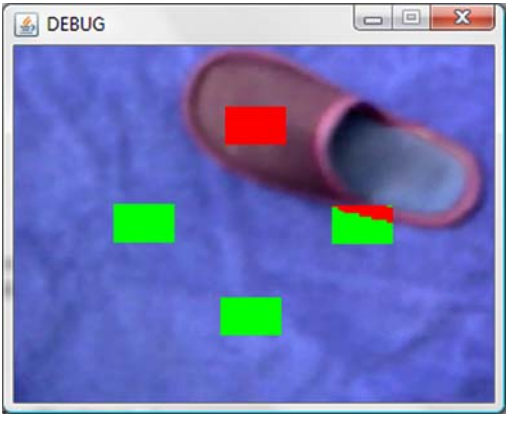
Algoritem za določitev akcij v sami igri temelji na odštevanju dveh slik [8]. Sistem si ob inicializaciji zapomni prvo sliko iz vhodne enote in jo registrira kot referenčno sliko. Ta slika potem služi za primerjavo z nadaljnjimi slikami. Na tem mestu postavimo pravilo, da uporabnik ob inicializaciji sistema ne sme prekrivati igralne plošče. Koraki algoritma:

1. Določitev regij zanimanja.
2. Zajem referenčne slike.
3. Zajem vhodne slike.
4. Primerjava HSL vrednosti slikovnih elementov trenutne in referenčne slike v vseh regijah zanimanja.
5. Odločanje, ali uporabnik stoji na kateri izmed komponent igralne plošče.
6. Ponovitev korakov 3-6.

Prvi in drugi korak se izvedeta izključno ob inicializaciji sistema. Vsi nadaljnji koraki se izvajajo ob vsakem zajemu slike iz vhodne enote. Regije zanimanja predstavljajo posamezne komponente igralne plošče. V tej verziji platforme jih določamo ročno - vnesemo koordinate komponent igralne plošče.

Primerjavo vrednosti dveh slikovnih elementov izvajamo v HSL barvnem prostoru [8]. Razlog za to je velika imunost HSL barvnega prostora na spremembe v osvetlitvi. Izkaže se, da za uspešno primerjavo dveh slikovnih elementov ne potrebujemo vseh komponent HSL barvnega prostora. Dovolj je že primerjava vrednosti Hue komponente. Omejitev, ki nastopi z uporabo tega barvnega prostora, je zahteva po živih barvah komponent igralne plošče, saj je prostor pri sivinah nestabilen.

V četrtem koraku izračunamo absolutno vrednost razlike slikovnih elementov trenutne in referenčne slike. Razliko registriramo kot spremembo, če je večja od vrednosti tolerance *K*. Na podlagi testiranja smo ugotovili, da je primerna vrednost za toleranco *K* enaka 10. Na sliki 3 je predstavljeno razhroščevalno okno enote za procesiranje.




Slika 3: Razhroščevalno okno enote za procesiranje; z rdečimi slikovnimi elementi so predstavljene razlike v regijah zanimanja med trenutno in referenčno sliko.

Odločanje v petem koraku temelji na preštevanju spremenjenih slikovnih elementov znotraj posamezne regije zanimanja. Uporabnik stoji na komponenti igralne plošče, če je ta vsota večja od danega koeficienta *S*. Za koeficient *S* smo uporabili polovico vseh slikovnih elementov v regiji zanimanja.

## 2.3 Razprava

Predstavljena implementacija nudi solidno uporabniško izkušnjo. Platforma se odziva na premike uporabnika in hkrati deluje imuno pri spremembah lokalne osvetlitve (npr. sencah). Hitrost procesiranja slike je v našem primeru omejena z uporabljeno enoto za zajem slike [9], ki omogoča zajem 30 slik na sekundo. Meritve so pokazale, da predstavljeni algoritem omogoča procesiranje do 50 slik na sekundo. Demonstracijska postavitev labirinta je predstavljen na sliki 4.

Slika 4: Demonstracijska postavitev labirinta.

Nadaljnje delo zajema samodejno detekcijo komponent igralne plošče, zaznavanje sprememb v globalni osvetlitvi in izpopolnitev grafike same igre na izhodni enoti. Prav tako lahko uvedemo različne scenarije igranja, ki bi dodatno popestrili igro. V tej smeri je še veliko neizkoriščenega prostora, za odkrivanje katerega nam predstavlja platforma dobro izhodišče.

# 3. MEHURČKI

## 3.1 Opis

Tukaj je bil naš cilj razviti program, ki prikazuje sliko iz kamere, v ospredju pa se dvigujejo milni mehurčki, ki ob sovpadanju z roko oziroma prstom počijo. Igra je narejena v programskem jeziku Java, za komunikacijo s kamero pa se uporablja njen paket za delo z multimediji JMF [4].

### *3.1.1 Prepoznava roke*

Za prepoznavo roke smo uporabili naslednje predpostavke oziroma lastnosti:

1. Roka je premikajoča se stvar in zato je potrebno iz slike odstraniti vse druge premikajoče se dele, na primer glavo. Na sliki se locira obraz, okoli obraza se poišče rob med glavo in ozadjem, regija pa mora biti v obliki elipse. Tako je označena regija, ki ima enako barvo kot roka in predstavlja pomembno informacijo za nadaljnje procesiranje.
2. V primeru, da na sliki najdemo roko, obstaja možnost, da jo na sliki iščemo nekajkrat na sekundo od začetka do konca slike, po vsakem slikovnem elementu posebej, ali pa, da si program zapomni prejšnji položaj roke. V slednjem primeru lahko sklepamo, da se položaj roke v majhnem delu sekunde ne bo drastično spremenil. Tako lahko iščemo roko v omejeni regiji, kar zelo razbremeni sam algoritem oziroma zmanjša porabo procesorskega časa. V primeru, da v tej regiji ne najdemo roke, pa lahko spet uporabimo metodo iskanja roke na celi sliki.
3. Roka se bo, ko bo iskala mehurčke, skoraj zagotovo bolj premikala od ostalih delov telesa. Število spremenjenih slikovnih elementov bo tako večje okoli roke, kot pa okoli glave.
4. Ob premiku prsta je razlika med dvema zaporednima slikama predstavljena v obliki pravokotnikov in naša naloga je poiskati te pravokotnike, ker s tem poiščemo prst. Slika 5 prikazuje kako smo s polaganjem zelenega pravokotnika na sliko razlik poiskali prst.


Slika 5: Zaznava gibanja.

## 3.2 Implementacija

Kreiranje grafičnega vmesnika in potrebnih niti je v Javi dokaj preprosto, zavedati se je treba le, da je izvajanje same Jave na računalniku nekoliko počasnejše od izvajanje programov napisanih na primer v C++.

Da bi program še vseeno deloval hitro in lahko izračunal vse potrebne izračune, smo izbrali ločljivost 320×240 slikovnih elementov. Za delo z barvami smo uporabili RGB barvni prostor.

### *3.2.1 Algoritem*

Osnovni potek delovanja:

1. Program zajame barvno sliko iz kamere.
2. Program zajame novo sliko ter jo odšteje od stare. Z odštevanjem obeh slik na njih najdemo premike. Za vsak slikovni element posebej najdemo primer, ko je rezultat odštevanja 0 oziroma čim bližje 0. V teh primerih se na tej lokaciji ni zgodilo nič. Ta slikovni element pobarvamo s črno. Večja kot je razlika (vzamemo absolutno vrednost rezultata), večja je verjetnost, da se je na tem področju nekaj spremenilo. Ta slikovni element obarvamo z belo. Mi smo prag za odločitev izkustveno nastavili na vrednost 50.
3. Iz množice črnih in belih slikovnih elementov sestavimo sliko razlik, na kateri se zelo lepo vidijo premiki rok in glave.
4. Roko poiščemo tako, da se v zanki sprehodimo po vseh slikovnih elementih, zraven upoštevamo še sosednje slikovne elemente v soseščini 5×4, ki skupaj sestavljajo pravokotnik. Če število belih slikovnih elementov zadošča pogoju, da je v pravokotniku vsaj 70% belih slikovnih elementov, potem predpostavljamo, da se na tem mestu nahaja prst. Z iskanjem končamo takoj, ko prvič zadostimo pogoju.
5. V zgornji plasti slike se gibljejo mehurčki, ki so definirani z radijem in v primeru, da je konica prsta od mehurčka oddaljena za manj kot njegov radij (v našem primeru 15 slikovnih elementov), mehurček poči (slika 6).
6. Ob vsaki osvežitvi pogledamo, koliko mehurčkov, ki jih definiramo kot niti, je že prišlo do vrha in tako nit odstranimo.
7. Vsak mehurček ob pojavitvi prevzame eno od vnaprej pripravljenih slikic milnih mehurčkov. Sličice so shranjene v .png formatu.

## 3.3 Razprava

Glede na število operacij, ki jih mora izvesti sam program, je uporabniška izkušnja odvisna od strojne opreme računalnika. Delovanje samega programa je tudi zelo odvisno od nastavitev kamere. Pri zajemu slike bi bilo potrebno na dobljeno sliko za obdelavo postaviti nekaj filtrov, da bi bili rezultati obdelave karseda optimalni.

Zelo velik problem predstavlja primer, kjer je barva ozadja slike zelo podobna barvi kože. V takem primeru se pri samem iskanju gibanja ne bo našlo dovolj razlik med slikama.

Sama zaznava roke deluje presenetljivo dobro, vendar je to vseeno največja pomanjkljivost samega programa. Tako je potrebno poiskati boljšo metodo prepoznave roke.

Več interaktivnosti in efektov bi lahko uporabil tudi pri pokanju samih mehurčkov. Ob dotiku prsta, bi se recimo lahko zgodila animacija, ki bi dodatno popestrila sam program.


Slika 6: Delujoč program.

# 4. NAVIDEZNI SLIKAR

## 4.1 Opis

Naš cilj je bil razviti preprost javanski program, ki bi omogočal slikanje zgolj z uporabo kamere. Program bi tako s pomočjo kamere zajemal slike premikajočih se rok uporabnika, nato pa bi zajete slike ustrezno obdelal in na podlagi tega izrisoval sliko v realnem času. Osnova za delovanje je tako nameščena Java, JMF [4] in ustrezna kamera (podpora le teh je nekoliko omejena, saj se JMF trenutno žal ne posodablja več).

## 4.2 Implementacija

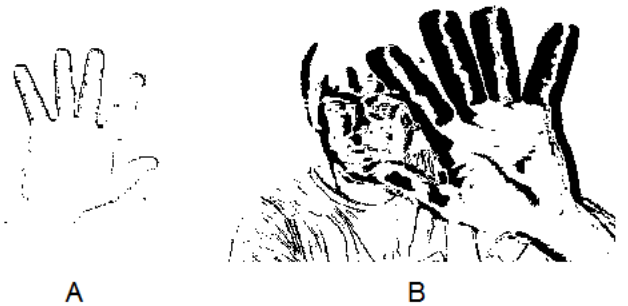
### *4.2.1 Algoritem*

Osnovni koraki algoritma so:

1. Postopek začnemo z zajemom slike. Ker pa je obdelava barvne slike in iskanje posebnosti (v tem primeru dlani oz. posameznih prstov) lahko zelo zamudno opravilo, smo se že v osnovi odločili za analiziranje razlik dveh časovno zaporedno zajetih slik. Pri tem sicer izgubimo barvno informacijo, ki bi nam

morebiti koristila pri prepoznavi dlani, vendar če pri tem upoštevamo, da vsi ljudje vendarle nimamo enake barve kože in da se osvetlitev lahko stalno spreminja, hitro vidimo, da je nadaljnja analiza, zgolj na podlagi razlik, veliko enostavnejša. Še posebej, ker s tem elegantno izločimo vplive osvetlitve, sama aplikacija pa je tako veliko bolj robustna.

2. Idealen čas med zajemom zaporednih slik je spremenljiv glede na okoliščine. Če nastavimo premajhno vrednost, morebiti sploh ne bomo zaznali uporabnikovih premikov, če pa je ta prevelika pa lahko pride do prevelikih popačenj. Prav tako nas lahko motijo sence, spreminjanje osvetlitve, saj se poveča število zaznav nezaželenih sprememb. Vsekakor pa vrednost ne more biti, oziroma je nesmiselno, manjša od 34 ms, ki jih potrebuje kamera za zajem naslednje slike. Privzeta vrednost je 40 ms.
3. Dobljeni barvni sliki primerjamo za vsak slikovni element posebej, glede na izkustveno določen prag vrednosti. Če je odstopanje pri posameznem slikovnem elementu nad pragom, si za tisto mesto shranimo spremembo. Nastala slika je zgolj binarna slika, s katere lahko jasno vidimo obrise premikajočih se stvari (slika 7).
4. Na dobljeni binarni sliki razlik poiščemo največje spremenjeno območje (slika 8) in sprožimo ustrezno akcijo risanja.




Slika 7: Binarna slika razlik pri premikanju (a) zgolj dlani in (b) celotnega telesa.

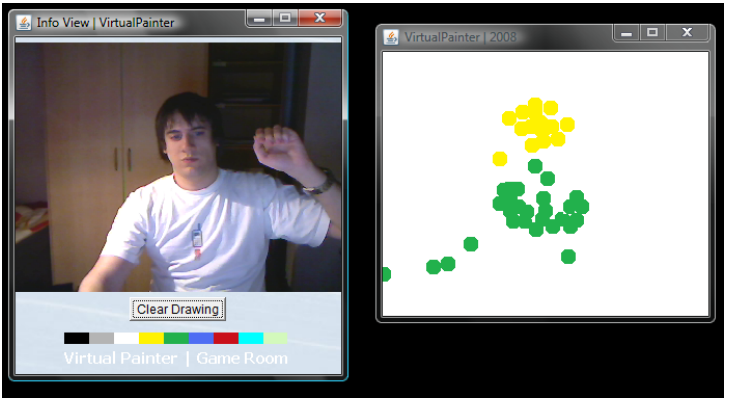
Na sliki 7a lahko jasno razločimo prste, prav tako na sliki ni praktično nobenih večjih motenj. Zaradi tega je prvotna ideja, da bi se na sliki zgolj iskalo navpične zvezne linije na prvi pogled skoraj popolna. S tem bi lahko tudi natančneje analizirali sam položaj dlani, npr. široko razprto, zaprto dlan. S pomočjo teh podatkov bi se dalo na relativno preprost način implementirati menjanje barve ali širine čopiča z gibi rok, kar bi bil odličen dodatek Virtualnemu slikarju. Ker pa je resnični svet drugačen od zamišljenega, se zadeve lahko hitro zapletejo. Uporabnik namreč lahko premika tudi glavo in telo (kot je razvidno iz slike 7b), za njim se lahko pojavijo drugi ljudje, za katere bi po algoritmu lahko napačno sklepali, da so prsti! Zaradi te kompleksnosti smo se začasno odločili zavreči idejo iskanja posameznih prstov oz. vzdolžnih linij le-teh.

Slika 8: Prvo največje spremenjeno območje.

### *4.2.2 Izvedba*

Po večkratnem testiranju smo ugotovili, da je preprosto pravzaprav najboljše. Znebili smo se vseh podrobnejših analiz in se zgolj usmerili v iskanje skupkov sprememb. Torej zgolj nekega območja, kjer se nahaja določeno število spremenjenih slikovnih elementov. Da bi bila zadeva še preprostejša (program vendarle teče v Javi), preverjamo zgolj manjša kvadratna območja, podobno kot pri Mehurčkih. Pri prvem takem območju program javi, da je na tistem mestu objekt, ki se je očitno premaknil, zato sprožimo ustrezno akcijo risanja. Takšno iskanje je na prvi pogled zelo robato, vendar se hitro izkaže za zanesljivejše od posamičnega preverjanja linij prstov in je hkrati občutno hitrejše.

Uporabniški vmesnik je preprost in je sestavljen iz dveh oken (slika 9). Informacijskega okna, kjer se prikazuje zajeta slika velikosti 320×240 slikovnih elementov, in virtualnega platna, kjer se v realnem času prikazuje izdelek uporabnika. Nizka ločljivost 320×240 slikovnih elementov nudi hitro obdelavo, prav tako pa uporabnik premike svoje dlani jasno vidi. Akciji, ki zahtevata uporabo miši oziroma tipkovnice, sta izbira barve in brisanje platna, kontroli za ti opravili pa se nahajata v spodnjem delu informacijskega okna.




Slika 9: Uporabniški vmesnik.

### 4.3 Razprava

Navidezni slikar deluje dokaj hitro in brez zatikanj. Edina omejitev je hitrost kamere, ki lahko sliko zajame približno na vsakih 33 milisekund. Čas računanja je v primerjavi s tem zanemarljiv, program deluje že dovolj hitro, zato posebna optimizacija tukaj ni potrebna. Težave z risanjem nastopijo, če je uporabnik preblizu kamere, saj je dlan navidezno zelo velika, kar povzroči poskakovanje čopiča po površini dlani. Zaradi tega razloga in motenj, ki lahko vplivajo na delovanje programa, bi bilo potrebno analizirati samo dlan. S položajem prstov bi bilo tako preprosto upravljati z debelino čopiča ali menjanjem barv. Menjanje barv trenutno poteka prek standardnega vhoda, kar nekoliko pokvari uporabniško izkušnjo, zato bi bila izboljšava v tej smeri vsekakor dobrodošla.

## 5. ZAKLJUČEK

Na treh konkretnih primerih smo pokazali, da gradnja enostavnih nekonvencionalnih uporabniških vmesnikov ni zapleten proces. Kljub temu, da smo uporabili razmeroma preproste algoritme, so sistemi odzivni in nudijo solidno uporabniško izkušnjo. Takšni uporabniški vmesniki igralcu približajo občutek realnosti v igri. Podan pristop je cenovno dostopen, saj kot dodatek potrebujemo le spletno kamero. Demonstracijske filme si bralec lahko ogleda na [10].

Če bi želeli zgraditi zapletene igre, bi potrebovali več različnih ukazov in bi morali uporabiti bolj kompleksne algoritme računalniškega vida. Naš namen je bil prikazati, da lahko igre krmilimo tudi s čim drugim kot s tipkovnico ali miško. Implementirane igre kažejo, da je to mogoče. Pri gradnji bolj kompleksnih iger smo omejeni samo z algoritmi prepoznavanja gibov človeškega telesa. Glede na to, da so ti algoritmi že kar precej izpopolnjeni, se lahko zgodi, da bomo takšne uporabniške vmesnike v prihodnosti redno srečevali pri računalniških igrah.

## LITERATURA


1. *http://en.wikipedia.org/wiki/Wii_Remote* − Wii Remote.
2. *http://www.apple.com/iphone/features/accelerometer.html* − Apple Accelerometer.
3. *http://en.wikipedia.org/wiki/PlayStation_Eye* − Playstation Eye.
4. *http://java.sun.com/javase/technologies/desktop/media/jmf/* − Java Media Framewok.
5. *http://www.flashsandy.org* − Sandy 3D engine.
6. *http://livedocs.adobe.com/flash/9.0/main/00000318.html* − Flash sockets.
7. *http://java.sun.com/docs/books/tutorial/networking/sockets/* − Java sockets.
8. L. G. Shapiro, G. C. Stockman (2001), *Computer Vision*, Prentice-Hall.
9. *http://www.logitech.com/index.cfm/webcam_communications/webcams/devices/4225&cl=roeu,en* − Uporabljena enota za zajem slike Logitech QuickCam S7500.
10. *http://www.lrv.fri.uni-lj.si/~peterp/GameTeam/* − Demo posnetki.