Բովանդակություն

Համառոտագիր

1-2 պարբերություն

Ներածություն

2-4 էջ

Գլուխ 1. Գրականության վերլուծական ակնարկ

1.1 Օբյեկտների ճանաչում

Օբյեկտների ճանաչման համակարգը գտնում է իրական աշխարհի օբյեկտներ րական աշխարհի պատկերներում՝ օգտագործելով օբյեկտի մոդելներ, որոնք կանխավ հայտնի են: Խնդիրը զարմանալիորեն դժվար է: Մարդիկ անընդհատ կատարում են օբյեկտների ճանաչման գործողություն առանց մեծ ջանքերի, այն դեպքում երբ այդ պնդրի ալգորիթմական նկարագրությունը եւ լուծումը համակարգչի միջոցով հանդիպում է շատ բարդույթների: Այս գլխում մենք ուսումնասիրվում են օբյեկների ճանաչման գործընթացը եւ ներկայացվում օբյեկտների ճանաչման լայն ասպարեզի կիրառական խնդիրների լուծման ընդհանանրացված մոտեցումներ: Կդիտարկվեն տարբեր տեսակի գոծույթներ, որ ճանաչողական համակարգը պետք է իրականացնի: Կուսումնասիրվեն այս գործույթների բարդությունները:

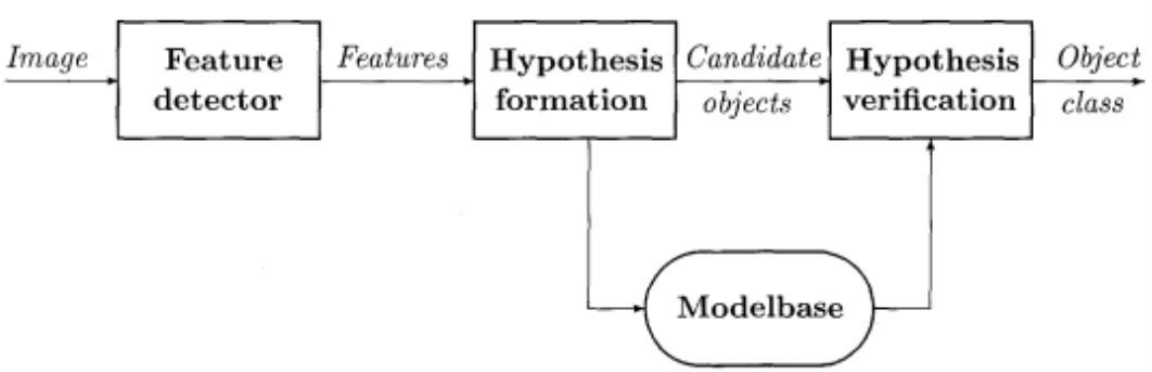
Օբյեկտների ճանաչման խնդրիը կարելի է սահմանել որպես պիտակավորման խնդիր՝ հիմնված նախապես հայտնի օբյեկտների մոդելների վրա: Ունենալով պատկեր, որը պարունակում է մեկ կամ մի քանի հետաքրքրության առարկա հանդիսացող օբյեկտներ եւ պիտակների հավաքածու, որոնք համապատասխանում են համակարգին հայտնի մոդելներին՝ համակարգը պետք է վերագրի ճիշտ պիտակը պատկերի համապատասխան ռեգիոնին կամ ռեգիոններին: Օբյեկտների ճանաչման խնդիրը սերտ կապված է պատկերի սեգմենտավորման խնդրի հետ:

Հաջորդող գլուխներում կդիտարկվեն օբյեկտների ճանաչման բազային հիմունքները: Կներկայացվեն օբյեկների ճանաչման համակարգի ճարտարապետությունը՝ իր տարրերով: Կքննարկվեն վերջիններիս դերը տարբեր տեսակի օբյեկների ճանաչման համակարգերում:

1.1.1 Համակարգի տարրերը

Օբյետկների ճանաչման համակարգի ընդհանրացված ճարտարապետությունը բաղկացած է 4 տարրերից

- Մոդելների հենք
- Հատկանիշների դետեկտոր
- Հիպոթեզի առաջարկման տարր
- Հիպոթեզի հաստատման տարր



Տարրեի փոխազդեցության բլոկ սխեման (նկ.1) ցույց է տալիս ինչ հերթականությամբ են տարրերը աշխատում:

Մոդելների հենքը պարունակում է համակարգին հայտնի բոլոր մոդելները: Մոդելի պարունակությունը կախված է ճանաչման խնդրի լուծման ընտրված մեթոդից: Այն կարող է տատանվել որակական կամ ֆունկցիոնալ նկարագրումից, մինչև օբյեկտի մակերևույթի ճշտգրիտ երկրաչափական նկարագրության: Շատ դեպքերում օբյեկտների մոդելները աբստրակտ հատկանիշների վեկտորներ են: Հատկանիշը օբյեկտի որոշակի բաղկացուցիչ է, որը կարևոր չափանիշ է այլ օբյեկտների հետ համեմատման դեպքում: Չափսը, գույնը և երկրաչափական ձևը հաճախակի օգտագործվող հատկանիշներ են:

Հատկանիշների դետեկտորը պատկերի վրա կիրառում է օպերատորներ և իդենտիֆիկացնում է հատկանիշների երկրաչափական դիրքերը՝ օգնելով կազմել օբյեկտի հիպոթեզ: Համակարգի կողմից օգտագործվող հատկանիշները կախված են ճանաչման ենթակա օբյետկենրի տեսակներից, ինչպես նաև մոդելների հենքի կառուցվածքից: Օգտվելով հատկանիշների դետեկտորի արդյունքներից՝ հիպոթեզ առաջարկող տարրը վերագրում է հավանականություններ այն դասի օբյետկներին, որոնց գոյությունը պատկերում առավել հավանական է: Այս քայլը օգտագործվում է փնտրման սահմանները փոքրացնելու նպատակով: Մոդելի հենքը կառուցվածքում առկա է ինդեքսավորման համակարգ, որը թույլ է տալիս արագացնել քիչ հավանական օբյեկտների հեռացումը հիպոթեզի կազմման համար: Հաստատող տարրը այնուհետև վերահաստատում է հիպոթեզը և ճշտգրտում նախնական հավանականությունները: Վերջիվերջո համակարգը ընտրում է ամենաբարձր հավանականություն ունեցող օբյեկտը՝ հիմնվելով բոլոր հավաքված փաստերի վրա:

Օբյետկների ճանաչման բոլոր համակարգերը օգտագործում են մոդելներ բացահայտ կամ ոչ բացահայտ կերպով և պարունակում են հատկանիշների դետեկտորներ հիմնված այդ մոդելների վրա: Հիպոթեզի առաջարկման և հաստատման տարրերի կարևորությունը կախված ճանաչման մեթոդի ընտրությունից կարող են տարբերվել: Որոշ համակարգեր օգտագործում են միմիայն հիպոթեզի առաջարկման տարրը և ընտրում այդ տարրի մշակման արդյունքում մեծագույն հավանականությունը ունեցող օբյեկտը, որպես ճիշտ: Օրինաչափությունների դասակարգում կատարող համակարգերը այսպիսի մոտեցման լավ օրինակ են: Շատ արհեստական բանականային համակարգեր, ընդհակառակը փոքր կարևորություն են տալիս հիպոթեզի փուլի արդյուներին և հիմնականում հենվում են

հաստատման փուլի արդյունքների վրա: Ավելին, օբյեկտների ճանաչման դասական համարվող մոտեցումներից կաղապարային համապատասխանեցումը՝ ընդհանրապես շրջանցում է հիպոթեզի առաջարկման փուլը:

Օբյեկտների ճանաչման համակարգը պետք է ընտրի վերը նշված տարրերի իրականացման համար համապատասխան միջոցներ և գործիքներ: Մեթոդների ընտրությունը առավելապես կախված է կոնկրետ կիրառումից: Ստորև նկարագրված են այն հիմնական խնդիրներըմ որոնք անհրաժեշտ է հաշվի առնել համակարգի կոնկրետ նախագծման համար:

Օբյեկտի կամ մոդելի ներկայացումը: Ինչպե՞ս ներկայացնել օբյեկտները մոդելների հենքում: Օբյեկտի ո՞ր հիմնական հատկանիշներն է անհրաժեշտ պահել այն մոդելներում: Օբյետկների, որոշակի դասերի համար երկրաչափական նկարագրությունը կարող է հասանելի և էֆֆեկտիվ լինել, այն դեպքում երբ մյուսների համար ստիպված ենք լինում բավարարվել ընդհանուր կամ ֆունկցիոնալ հատկանիշների վրա: Օբյեկնտի նկարագրությունը պետք է լինի սպառիչ, սակայն առանց տեղեկութային ավելորդությունների, պետք է կազմակերպված լինի այնպես, որ հեշտ հասանելի լինի ճանաչման համակարգի ցանկացած տարրի համար: Պատկերի բնույթից կախված, տարբեր հատկանիշների բացահայտումը պահանջում է տարբեր քանակության հաշվարկային հզորություն:

Հատկանիշների բացահայտում: Ո՞ր հատկանիշներն է անհրաժեշտ դիտարկել և ինչպե՞ս գտնել դրանք: Հատկանիշների մեծամասնությունը փնտրվում է երկչափ նկարում, սակայն իրենից ներկայացնում է իրական աշխարհի եռաչափ մարմին:

Հատկանիշ-մոդել համապատասխանեցումը: Ինչպե՞ս է կարելի պատկերից բացահայտված հատկանիշները համապատասխանեցնել մոդելի պարունակության հետ: Օբյեկների ճանաչման շատ խնդիրներում դիտարկվում են բազմաթիվ հատկանիշներ ու օբյեկտների դասեր: Դրանց՝ հատարկման եղանակով համապատասխանեցումը, կարող է չափազանց դանդաղ լինել կիրառական բնույթ ունենալու համար: Հատկանիշների էֆֆեկտիվությունը և դրանց՝ մոդելին համապատասխանեցման արագագործությունը, անհրաշեշտ է հաշվի առնել՝ համակարգը մշալելիս:

Հիպոթեզի կազմավորումը: Ինչպե՞ս է կարելի ընտրել հավանական օբյեկտների հավաքածու՝ հիմնվելով բացահայտված հատկանիշների վրա: Ինպե՞ս այդ գտնված օբյետկներին վերագրել համապատասխանելիության հավանականություն: Հիպոթեզի առաջարկման փուլը, փաստացի, միջոց է հնարավոր տարբերակների փնտրման սահմանները նեղացնելու համար: Այս քայլը օգտագործում է տվյալ խնդրի կիրառական բնույթի առանձնահատկությունների մասին տեղեկույթ, որպեսզի օբյեկնտերի հնարավոր բոլոր դասերից որոշներին դարձնի հավանական և զտի մնացածը: Դա կատարվում է հավանական դասերի օբյեկտներին համեմատական բարձր հավանականություններ վերագրելով: Այս չափանիշը ցույց է տալիս գտնված հատկանիշների պարագայում տվյալ օբյետկի առկայության հավանականությունը:

Օբյեկտի հաստատում: Ինչպե՞ս օգտագործել օբյեկտի մոդելները հնարավոր օբյեկտներից ամենահավանականը ընտրելու համար: Ամեն մի տեսակի օբյեկտի առկայությունը կարելի է ստուգել այդ օբյետկի մոդելի միջոցով: Պետք է դիտարկվեն բոլոր հնարավոր հիպոթեզները տվյալ օբյեկտի առկայությունը ստուգելու համար: Եթե մոդելը երկրաչափական է, հեշտ է ճշտգրիտ հաստատել այդ օբյեկտի առկայությունը՝ օգտագործելով տեսախցիկի դիրքը և տեսարանի այլ պարամետրեր: Այլ դեպքերում, օբյետկի առկայության հաստատումը կարող է լինել բարդ, իսկ որոշ դեպքերում և անհնարին:

Կախված խնդրի բարդությունից, նկ. 1-ում պատկերված սխեմայի մեկ կամ մի քանի տարրեր կարող են դառնալ տրիվիալ: Օրինակ՝ օրինաչափությունների ճանաչման վրա հիմնված համակարգերը չեն օգտագործում հատկանիշ-մոդել համապատասխանեցման կամ օբյետկների հաստատման փուլերը, փոխարենը անմիջապես օբյեկտներին վերագրում են հավանականություններ և ընտրում ամենամեծ հավանականությամբ օբյեկտը:

1.1.2 Օբյեկտների ճանաչման խնդրի բարդությունները

Պատկերում նկարված տեսարանը կախված է լուսավորությունից, տեսախցիկի պարամետրերից և տեսախցիկի դիրքից: Քանի որ օբյեկտը պետք է ճանաչվի բազմաթիվ այլ օբյետկներ պարունակող պատկերում, օբյեկտների ճանաչման խնդիրը կախված է մի քանի գործոններից:

Տեսարանի կայունությունը: Ճանաչման խնդրի բարդությունը կախված է նրանից, արդյոք մշակվող պատկերում օբյեկտի վրա ազդում են միևնույն արտաքին գործոնները՝ լուսավորվածություն, հետին պլան, տեսախցիկի պարամետրեր, տեսախցիկի դիտակետ, ինչ մոդելներում: Նշված գործոնների հավանական առանձնահատկությունները պետք է հաշվի առնվեն տրված կիրառման շրջանակներում էֆֆեկտիվ հատկանիշների ընտրության համար:

Պատկերի և մոդելների տարածությունները: Որոշ կիրառումներում դիտարկվող եռաչափ օբյեկտները կարող են դիտարկվել, որպես երկչափ: Այդպիսի դեպքերում մոդելները կարող են նկարագրվել երկչափ հատկորոշիչներով: Եթե մոդելներն երկչափ են և պերսպեկտիվ պրոյեկտման արդյունքը անհնար է անտեսել, իրավիճակը բարդանում է: Այս դեպքերում հատկանիշները բացահատվում են երկչափ տարածությունում, այն դեպքում երբ օբետկների մոդելները նկարագրվում են եռաչափ տարածությունում: Այսպիսով եռաչափ հատկանիշը կարող է թվալ այլ հատկանիշ երկչափ պատկերում: Այս խնդիրն կարող է առաջանալ նաև դինամիկ պատկեններում, որտեղ օբյեկտները շարժվում են:

Մոդելների հենքում օբյեկտների քանակը: Եթե մոդելների հենքում օբյետկների քանակը համեմատաբար փոքր է, իմաստալից է հրաժարվել հիպոթեզի առաջարկման փուլից: Հաջորդական համապատասխանեցումը կարող է ընդունելի լինել: Հիպոթեզի առաջարկման փուլի կարևորությունը բարձրանում է մոդելների հենքում օբյետկների քանակի աճին զուգընթաց: Համապատասխան հատկանիշների ընտրության խնդիրն ևս բարդանում է հենքում օբյետկների քանակի աճի հետ:

Պատկերում օբյեկտների քանակը և հնարավոր ծածկումները: Եթե պատկերում կա մեկ օբյեկտ, ապա այն կարող է տեսանելի լինել ամբողջությամբ: Պատկերում օբյեկտների քանակի աճին զուգընթաց բարձրանում է օբյեկտների ծածկումների հավանականությունը: Ծածկումը տարրական պատկերների մշակման հաշվարկների համար բավական լուրջ խնդիր է: Ծածկման արդյունքում կորում են որոշակի սպասվող հատկանիշներ և դրանց փոխարեն հայտնվում են անսպասելիները: Ծածկումը պետք է նաև հաշվի առնել հաստատման փուլում: Ընհանրապես պատկերներում օբյետկների ճանաչման խնդիրները բարդանում են օբյեկտների քանակի աճելուն զուգընթաց: Պատկերների սեգմենտավորման խնդրի բարդությունը հիմնականում պայմանավորված է օբյեկտների վերածածկումներով:

Կախված օբյետկների ճանաչման խնդրի վրա ազդող նշված գործոններից, մեթոդները կարելի է բաժանել հետևյալ 3 դասերի:

Երկչափ: Բազմաթիվ կիրառումներում պատկերները նկարահանվում են այնքան մեծ հեռավորությունից, որ կարելի է ընդունել, որ պրոյեկցիան օրթոգրաֆիկ է: Եթե օբյետկտկները տեսարանում միշտ գտնվում են միևնույն դիրքում, դրանք կարող են համարվել երկչափանի: Այս դեպքերում կարելի է օգտագործել երկչափանի մոլելներ: Այս դեպքում օբյեկտները կամ կլինեն ծածկված կամ չեն լինի:

Եռաչափ: Եթե հետաքրքրող օբյետկների պատկերները կարող են նկարահանված լինել պատահական դիտակետերից, ապա նույն օբյեկտը կունենա տարբեր տեսքեր տարբեր նկարներում: Եռաչափ օբյետկների մոդելներ օգտագործող օբյետկների ճանաչման համակարգերում անհրաժեշտ է հաշվի առնել պերսպեկտիվ էֆեկտը և տեսարանի դիտակետը:

Սեգմենտավորված: Այս մեթոդը ենթադրում է, որ պատկերները սեգմենտավորվում են՝ տարանջատելով օբյեկտները հետին պլանից:

1.1.3 Հատկանիշների հայտնաբերումը

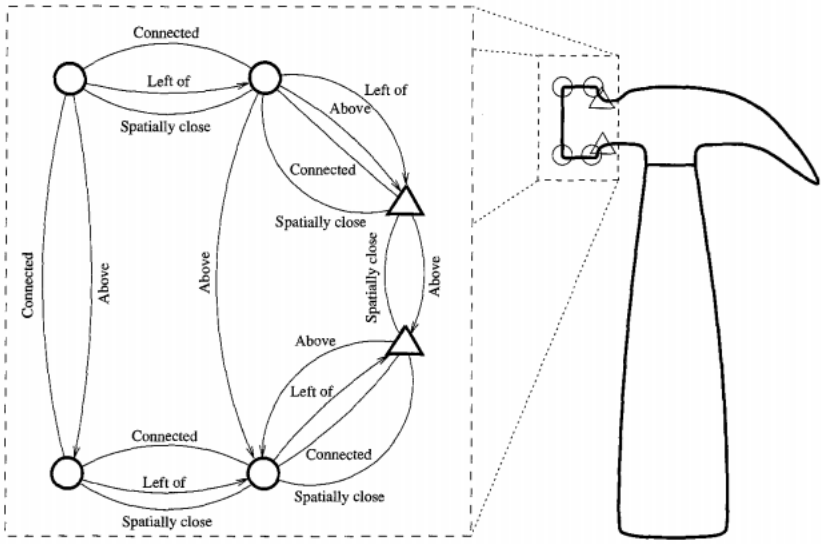
Օբյեկտների ճանաչման խնդիրներում օգտագործվում են տարատեսակ հատկանիշներ: Դրանց մեծ մասը հիմնված են պատկերի ռեգիոնների վրա: Ենթադրվում է, որ փակ կոնտուրով ռեգիոնները իրենցից ներկայացնում են կամ օբյեկտ կամ օբյեկտի որևէ հատված: Ստորև կներկայացվեն ամենատարածված հատկանիշները:

Գլոբալ հատկանիշներ: Գլոբալ հատկանիշներ եմ պատկերում ռեգիոնների մակերեսը, տրամագիծը, Ֆուրյեյի նկարագրիչները, մոմենտը և այլն: Գլոբալ հատկանիշները կարող են հաշվարկվել կամ ռեգիոնի ներքին կետերի համար, կամ էլ եզրագծի կետերի համար: Բոլոր դեպքերում նպատակն է գտնել այնպիսի նկարագրիչներ, որոնք հաշվարկվում են հաշվի առնելով բոլոր կետերը, դրանց դիրքերը, տարածական փոխհարաբերությունները և վառությունը:

Լոկալ հատկանիշներ: Լոկալ հատկանիշները հիմնականում վերաբերվում են ռեգիոնի եզրագծին կամ բավականաչափ փոքր ռեգիոնին: Կորությունը և նման այլ հատկանիշներ հաճախակի օգտագործվում են, որպես լոկալ հատկանիշ: Կորությունը կարող է վերաբերվել ինչպես ռեգիոնի եզրագծին այնպես էլ մակերևույթին: Մակերևույթը կարող է լինել վառության

մակերևույթը: Մեծ կորություն ունեցող կետերը կոչվում են անկյուններ և կարևոր դեր են խաղում օբյեկտների ճանաչման խնդրում: Լոկալ հատկանիշները կարող են պարունակել եզրագրի կոնկրետ տեսքի հատված: Տարածված լոկալ հատկանիշներ են կորությունը, եզրագծի սեգմենտները և անկյունները:

Հարաբերական հատկանիշներ: Հարաբերական հատկանիշները տարբեր գոյերի՝ ռեգիոնների, փակ կոնտուրների կամ լոկալ հատկանիշների, հարաբերական դիրքերով միմյանց նկատմամբ: Այս հատկանիշները հիմնականում ներառում են հատկանիշների միջև հեռավորությունը կամ հարաբերական ուղղվածությունը որպես չափանիշ: Այս հատկանիշները օգտակար են կոմպոզիտային օբյեկտների նկարագրման ժամանակ: Բարդ օբյեկտը նկարագրվում է իր պարզ ռեգիոնների հատկանիշներով և դրանց փոխհարաբերությամբ: Միևնույն հատկանիշը տարբեր փոխհարաբերություններում կարող է վկայել տարբեր օբյեկտների առկայության մասին:



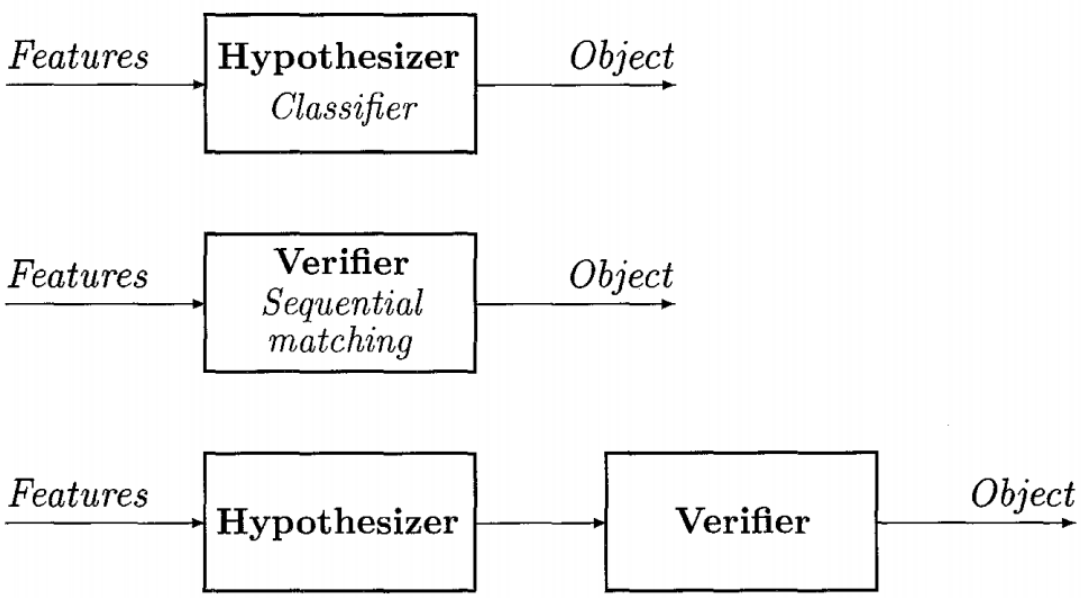
Number of Regions	2
Area	42574 pixels
Perimeter	1343 pixels

Նկ. 2-ում պատկերված է օբյեկտ իր հատկանիշներով: Այս օբյեկտը նկարագրելու համար օգտագործվում են թե լոկալ և թե գլոբալ հատկանիշներ: Հատկանիշների միջև հարաբերությունները ստեղծում են բաղադրյալ հատկանիշներ:

1.1.4 Ճանաչման մարտավարություններ

Օբյեկտի ճանաչումը քայլերի հերթականություն է, որը պետք է իրականացնել համապատասխան հատկանիշների հայտնաբերումից հետո: Ինչպես ավելի վաղ էր նշվել հատկանիշների հայտնաբերելուց հետո կազմվում է հիպոթեզ պատկերում հնարավոր օբյեկտների ներկայության մասին: Այս հիպոթեզը պետք է հաստատվի օբյեկտների մոդելների օգնությամբ: Ոչ բոլոր ճանաչման մեթոդներն են պահանջում ուժեղ հիպոթեզի կամ հաստատման փուլ: Ճանաչման ալգորիթմների մեծամասնությունը զարգացման ընթացքում

սկսել են օգտագործել այս երկու քայլները համակցված և տարբեր քանակությամբ: Ինչպես երևում է նկ. 3-ից նշված քայլերն կարող են օգտագործվել տարբեր համադրություններով:



Նկ. 3 Կախված խնդրի բարդությունից, ճանաչման մարտավարությունը կարող է օգտագործել հիպոթոզի առաջադրման և հաստատման փուլերն առանձին-առանձին կամ համակցված

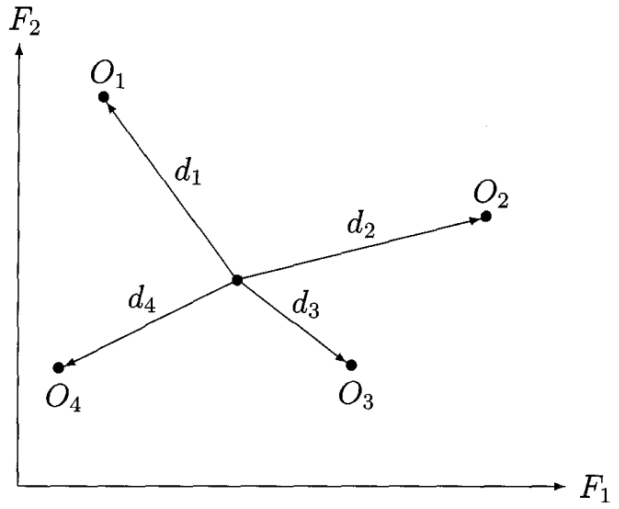
Հաջորդիվ դիտարկվում են օբյեկտների ճանաչման մի քանի հիմնական մարտավարություններ:

1.1.4.1Դասակարգում

Դասակարգման հիմնական միտքը կայանում է օբյեկտների ճանաչումը հիմնված դրանց հատկանիշների վրա: Օրինաչափությունների ճանաչման ինչպես նաև նեյտրոնային ցանցերի վրա հիմնված ճանաչման մեթոդները ընկնում են այս դասի մեջ: Այս բաժնում ներկայացված են մի քանի լայն օգտագործման դասակարգման մոտեցումներ: Բոլորն ենթադրում են, որ պատկերից բացահայտվել է N քանակի հատկանիշներ և որ վերջիններս նորմալիզացված են և կարող են ներկայացվել միևնույն հաշվարկման համակարգում: Հաջորդիվ ենթադրում ենք, որ օբյեկտի հատկանիշները կարող են ներկայացվել N-չափանի տարածությունում կետի տեսքով:

Ամենամոտ հարևանի դասակարգիչներ: Ենթադրենք, որ մոդելի օբյեկտը (հատկանիշների իդեալական արժեքների համակցությունը) ամեն մի դասի համար հայտնի է և ներկայացված է i-րդ դասի համար, որպես $f_{ij}, j = 1, \dots, N, i = 1, \dots, M$, որտեղ M-ը օբյեկտների դասերի քանակն է: Այժմ ենթադրենք, որ մենք փնտրում ենք U անհայտ օբյեկտը և գտել ենք վերջինիս հատկանիշները՝ $u_j, j = 1, \dots, N$: Հատկանիշների երկչափ տարածության համար այս իրավիճակը պատկերված է նկ. 4-ում: Օբյեկտի դասը որոշելու համար մենք գտնում ենք բոլոր դասերի նմուշային օբյեկտների և U-ի միջև նմանությունը, հաշվելով հատկանիշների

տարածությունում դրանց համապատասխանող կետերի հեռավորությունը, և U-ն վերագրում ենք ամենամոտ գտնվող դասին:



Նկ. 4 Յուրաքանչյուր դասի նմուշները ներկայացվում են, որպես կետ հատկանիշների տարածությունում: Անծանոթ օբյեկտը դասակարգվում է ամենամոտ դասին՝ հաշվարկելով հեռավորությունը հատկանիշների տարածությունում:

Հեռավորության հաշվարկի համար կարելի է օգտագործել Էվկլիդյան բանաձևը կամ ցանկացած այլ հատկանիշների կշռված համակցություն: Ընդհանուր դեպքում j դասի անծանոթ օբյեկտի համար d_j հեռավարութոյնը հաշվարկում ենք (3) բանաձևով:

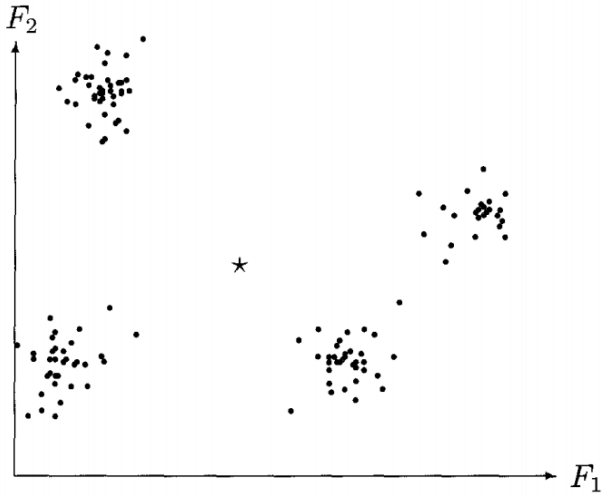
$$d_i = \left[\sum_{j=1}^{N}\left(u_j - f_{ij}\right)^2\right]^{\frac{1}{2}} \qquad (3)$$

Այնուհետև օբյեկտը վերագրվում է R դասին (4) բանաձևով:

$$d_R = \min_{1\le i\le M}[d_i] \qquad (4)$$

Վերջին մեթոդում հաշվարկվում էր անհայտ օբյեկտի հատկանիշների տարածությունում հեռավորությունը նմուշային կամ մոդելի օբյեկտից: Պրակտիկայում, սակայն, դժվար է ունենալ օբյեկտի նմուշային մոդել: Շատ օբյեկտներ կարող են պատկանել միևնույն դասին: Այս դեպքում պետք է հաշվի առնվեն տվյալ դասի բոլոր հայտնի օբյեկտների հատկանիշների արժեքները: Նման իրավիճակը ներկայացված է նկ. 5-ում: Տվյալ սցենարում հնարավոր է երկու մոտեցում՝

1. Համարել տվյալ դասի օբյեկտները ներկայացնող կետերի կլաստերի կենտրոնը, որպես նմուշային օբյեկտի կետ և հաշվել անհայտ օբյեկտի հեռավորությունը այդ կետներրոնից:
2. Անհայտ օբյեկտին դասել իրեն ամենամոտ գտնվող օբյեկտի դասին:



Նկ. 5 Մոդելի հենքի բոլոր օբյեկտները ներկայացվում են հատկանիշների տարածությունում կետի տեսքով: Ամեն մի դաս ներկայացնում է այդ կետերի կլաստեր: Անհայտ օբյեկտը որևէ դասին վերագրելու համար հաշվում ենք այդ օբյեկտն ներկայացնող կետի հեռավորությունը կլաստերների կենտրոններից կամ ամեն մի կլաստերի ամենամոտ կետերից: Ընտրվում է ամենամոտ կետը:

Բայեսյան դասակարգիչներ: Օբյեկտների ճանաչման Բայեսյան մեթոդը կիրառվում է, երբ հայտնի օբյեկտների բաշխումը նախորդ օրինակից ավելի խառն է: Նախորդ գլխում ենթադրվում էր, որ տարբեր դասերի օբյեկտները հատկանիշների տարածությունում զբաղեցնում են չհատվող իրարից հեռու տարածքներ՝ կազմելով առանձնացված կլաստերներ: Ընդհանուր դեպքում տարբեր օբյեկտների հատկանիշների արժեքները վերադրվում են: Հետևաբար, ինչպես ցուցադրված է նկ. 6 ա.-ի օրինակում(այստեղ դիտարկվում է միաչափ հատկանիշների տարածություն) մի քանի դասի օբյեկտներ կարող են ունենալ հատկանիշի միևնույն արժեքը: Երբ փորձենք հատկանիշների տարածությունից եզրակացնել անհայտ օբյեկտի դասը, կստանանք. որ մի քանի դասեր հավասարապես լավ թեկնածուներ են: Տվյալ իրավիճակներում, որոշում կայացնելու համար կարող է օգտագործվել Բայեսյան մոտեցումը:

Բայեսյան մոտեցման մեջ օգտագործվում է օբյեկտում հատկանիշի առկայության հավանականային գիտելիքները և տվյալ կիրառման մեջ տարբեր հայտնի օբյեկտների պատահման հաճախականությունը: Ենթադրենք, որ j դասի օբյեկտի համար պատահման հավանականությունը $P(\omega_j)$ է: Սա նշանակում է, որ մոդելին հայտնի բոլոր օբյեկտների համար գոյություն ունի պատահելու հայտնի հավանականություն և այլ օբյեկտի մասին տեղեկությունների նույնիսկ իսպառ բացակայության պարագայում կարելի է մինիմիզացնել որոշման սխալը՝ անհայտ օբյեկտը դասելով ամենից բարձր հավանականություն ունեցող դասին:

Օբյեկտի, որևէ դասին պատկանելու մասին որոշումները հիմնականում կայացվում են այդ օբյեկտի հատկանիշների դիտարկման արդյունքում: Մտցնենք պայմանական հավանականության գաղափարը և նշանակենք այն $p(x|\omega_j)$: Եթե ունենք հավանականային

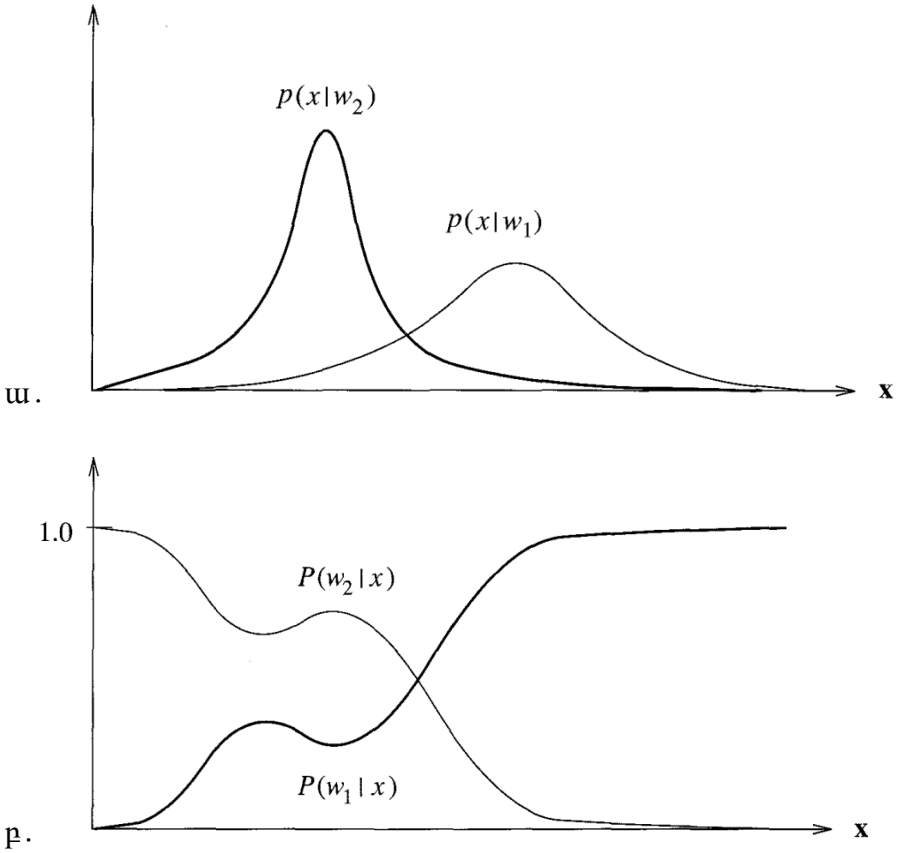
տեղեկատվություն մոդելին հայտնի օբյեկտների համար, այս մեծությունը ցույց է տալիս, որ հատկանիշի դիտարված x արժեքի համար անհայտ օբյեկտի j դասին պատկանելիության հավանականությունն է $p(x|\omega_j)$: Ունենալով այս տեղեկութույթը կարելի է հաշվել վերջնական հաեվանականությունը $p(\omega_j|x)$: Վերջինս ցույց է տալիս, ինչքանով է տվյալ դիտարկումներից ստացված տեղեկույթի հիման վրա հավանական, որ անհայտ օբյեկտը պատկանում է j դասին: Օգտագործելով Բայեսի օրենքը այս հավանականությունը տրվում է (4) բանաձևով:

$$P(\omega_j|x) = \frac{p(x|\omega_j)P(\omega_j)}{p(x)} \qquad (6)$$

$$p(x) = \sum_{j=1}^{N} p(x|\omega_j)P(\omega_j) \qquad (7)$$

Անհայտ օբյեկտը պետք է վերագրվի ամենահարձր վերջնական $P(\omega_j|x)$ հավանականությանը: Ինչպես երևում է նկ. 6 բ.-ում վերջնական հավանականությունը կախված է օբյեկտների մասին նախնական գիտելիքներից: Եթե այդ գիտելիքները փոխվում են, փոխվում է և հավանականությունը:

Մենք դիտարկեցինք Բայեսյան մոտեցումը մեկ հատկանիշի համար: Ընդհանուր դեպքում այն կարող է տարածվել բազմաթիվ հատկանիշներով տարածության վրա, որտեղ օգտագործվեն պայմանական խտության ֆունկցիաներ շատ հատկանիշների համար:



Նկ. 6 ա. Պայմանական խտության ֆունկցիան՝ $p(x|\omega_j)$ ցույց է տալիս հատկանիշի արժեքի հավանականությունը ամեն մի օբյեկտի դասի համար: բ. 2 նախնական հավանականությունների հիման վրա վերջնական հավանականություն

Նախապես կատարվող հաշվարկներ: Դասակարգման արդեն իսկ դիտարկված մոտեցումները օգտագործում են հատկանիշների տարածությունը և կախված հայտնի օբյեկտների հատկանիշների արժեքներից, այդ տարածությունը բաժանում են մասերի: Ամեն մի մասում հայտնվող օբյեկտը համարվում է այդ դասի օբյեկտ: Հատկանիշների տարածությունում ցանկացած կետի դաս վերագրելու համար բոլոր հաշվարկները կատարվում են կոնկրետ օբյետկի ճանաչման գործընթացից առաջ: Սա կոչվում է նախապես կատարվող հաշվարկներ: Վերջիններս նվազեցնում են կոնկրետ օբյեկտի ճանաչման գործընթացում ալգորիթմի աշխատանքի ժամանակը: Այս դեպքում ճանաչման գործընթացը վերածվում է տվյալների աղյուսակում փնտրման խնդրի, որի լուծման համար գոյություն ունեն շատ էֆֆեկտիվ ալգորիթմներ:

Նեյրոնային ցանց: Նեյրոնային ցանցերը լայն կիրառում են գտել օբյեկտների ճանաչման խնդիրներում: Դրանք իրականացնում են դասակարգում: Դրանց ուժեղ կողմը կայանում է հատկանիշների տարածությունը ոչ գծային սահմաններով բաժանելու ունակության մեջ: Այս սահմանները ստացվում են ցանցի ուսուցման արդյունքում: Ուսուցման շրջանում ճանաչման ենթակա շատ օբյետկներ են տրվում ցանցին: Եթե ուսուցման պատկերների հավաքույթն ընտրվի զգուշությամբ, որպեսզի ներկայացնի հետագայում ճանաչման ենթակա բոլոր օբյեկտները, ապա ցանցը կսովորի ճիշտ մասնատել հատկանիշների տարածությունը: Ճանաչման փուլում ցանցը աշխատում է այլ դասակարգիչների նման:

Նեյրոնային ցանցերի ամենամեծ առավելություններն են ոչ գծային սահմաններով հատկնաիշների տարածության բաժանումը և սովորելու հատկությունը: Հիմնական թերություններն են՝ կոնկրետ ճանաչման խնդրի պարագայում կիրառման մասին հավելյալ տեղեկատվության ներմուծելու անհնարությունը և դրանց մեջ սխալների փնտրման գործընթացի դժվարությունը:

1.1.4.2Համընկեցում

Դասակարգում կատարելու համար անհրաժեշտ է էֆֆեկտիվ հատկանիշների և խնդրի կիրառման մասին տեղեկույթ: Շատ կիրառումներում նախապես հայտնի չեն ոչ հատկանիշների հավանականությունները, ոչ էլ դասերի հավանականությունները, հետևաբար բացակայում է դասակարգիչ ձևավորելու համար անհրաժեշտ տեղեկույթը: Այսպիսի դեպքերում կարելի է անծանոթ օբյեկտը ուղղակիորեն համեմատել մոդելների օբյեկտների հետ և ընտրել ամենանման օբյեկտի դասը: Այս մոտեցումը օգտագործում է բոլոր հայտնի օբյեկտների մոդելները և ձևափոխում է մոդելի տեղեկույթը այնպիսի ձևաչափի, որ հնարավոր լինի համեմատել պատկերի պարունակության հետ՝ նմանությունը գտնելու նպատակով: Սա հիմնականում կատարվում է սեգմենտավորման գործընթացից հետո: Ստորև կքննարկվեն բազային համընկեցման մեթոդները:

Հատկանիշների համընկեցում: Ենթադրենք, որ ամեն մի օբյեկտի դաս ներկայացվում է իր հատկանիշներով: Նշանակենք i-րդ դասի օբյեկտի j-րդ հատկանիշի արժեքը f_{ij}: Անծանոթ օբյեկտի համար այդ հատկանիշները նշանակենք u_j-ով: i-րդ դասի օբյեկտի հետ նմանությունը տրվում է (7) բանաձևով, որտեղ ω_j-ն j-րդ հատկանիշի կշիռն է: Կշիռն ընտրվում է կախված հատկանիշի հարաբերական կարևորության: s_j-ն j-րդ հատկանիշի նմանությունն է: Սա կարող է լինել բացարձակ տարբերություն, նորմալիզացված տարբերություն կամ ցանկացած այլ հեռավորության չափ: Ամենից շատ օգտագործվում է բացարձակ տարբերությունը (8), որտեղ պետք է հաշվի առնել հատկանիշների կշիռների նորմալիզացումը:

$$S_j = \sum_{j=1}^{N} \omega_j s_j \quad (7)$$

$$s_j = |u_j - f_{ij}| \quad (8)$$

Օբյեկտը պիտակավորվում է k դասով, եթե S_k-ն նմանության ամենամեծ արժեքն է: Նկատենք, որ այս մոտեցման մեջ օգտագործվում են լոկալ կամ գլովար հատկանիշներ և չկան հարաբերականության հատկանիշներ:

Սիմվոլիկ համապատասխանեցում: Օբյեկտը կարող է ներկայացվել ոչ միայն իր հատկանիշներով, այլև այդ հատկանիշների միջև հարաբերություններով: Հարաբերությունները կարող են լինել տարածական կամ ցանկացած այլ տիպի: Այս դեպքում օբյեկտը կարելի է նկարագրել որպես գրաֆ: Գրաֆի ամեն մի գագաթ ներկայացնում է հատկանիշը, իսկ կողերը՝ դրանց միջև հարաբերությունները (նկ. 4): Օբյեկտի ճանաչման խնդիրը վերածվում է գրաֆների համապատասխանեցման խնդրի:

Գրաֆների համապատասխանեցման խնդիրը սահմանվում է հետևյալ կերպ: Ունենք G_1 և G_2, որոնց գագաթները նշանակենք N_{ij}, որտեղ i –ն գրաֆի համարն է, իսկ j-ն գագաթի համարն է: j և k գագաթների միջև հարաբերությունը ներկայացվում է R_{ijk}: Սահմանվում է գրաֆների նմանության չափանիշ, որը հաշվի է առնում բոլոր գագաթների և կողերի նմանությունը:

Օբյեկտների ճանաչման խնդիրների մեծամասնությունում ճանաչման ենթակա օբյեկտները տեսանելի են մասամբ: Ճանաչման համակարգը պետք է ճանաչի օբյեկտը ունենալով վերջինիս մասնակի պատկերը: Այն համակարգերը, որոնք օգտագործում են գլոբալ հատկանիշները և պետք է ուներ օբյեկտի բոլոր հատկանիշները այն ճանաչելու համար կիրառելի չեն այս դեպքերում: Պատկերում մասնակիորեն ներկա օբյեկտի ճանաչման խնդիրները նման են գրաֆների տեսությունում ուսումնասիրվող գրաֆների ներդրման խնդրին: Օբյեկտների ճանաչման խնդրի բնույթը փոխվում է, երբ դիտարկում ենք գագաթների միջև նմանությունը և փոխհարաբերությունները:

1.1.4.3Հատկանիշի ինդեքսավորում

1.1.4.4

1.1.5 Հաստատում

1.1.5.1 Կաղապարների համապատասխանեցում

1.2 Եզրերի փնտրման Քեննիի ալգորիթմը

Եզրերի փնտրումը՝ մաթեմատիկական մեթոդների համախումբ է, որի նպատակն է հայտնաբերել պատկերում վառության կտրուկ անցումներ՝ ընդհատումներ: Վառության կտրում անցումների կետերը կազմում են կորեր, որոնք կոչվում են եզր:

Դիտարկենք եզրերի փնտրման դասական ալգորիթմներից մեկը՝ Քեննիի ալգորիթմը: Այն բաղկացած է փուլերից.

- Գաուսյան ֆիլտրի միջոցով հարթեցնել պատկերը
- Գտնել պատկերի ինտենսիվության գրադիենտը
- Կիրառել ոչ մաքսիմալ ճնշում՝ եզրին չպատկանող փիքսելները հեռացնելու նպատակով
- Կիրառել կրկնակի շեմային ֆիլտրում
- Հեռացնել բոլոր այն թույլ եզրերը, որոնք չեն միացված ուժեղ եզրերին

Այժմ դիտարկենք ամեն մի փուլն ավելի մանրամասն:

1.2.1 Գաուսյան ֆիլտրի միջոցով պատկերի հարթեցում եւ աղմուկի վերացում

Աղմուկը հանգեցնում է պատկերում կեղծ եզրերի հայտնաբերմանը: Գաուսյան ֆիլտրը կիրառվում է աղմուկի վերացման համար: (2k + 1)(2k + 1) չափսի միջուկի Գաուսյան ֆիլտրի բանաձեւը տրվում է (1) արտահայտությամբ:

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-k-1)^2+(j-k-1)^2}{2\sigma^2}\right) \qquad (1)$$

Ալգորիթմի արագագործությունը կախված է միջուկի չափսից: Միջուկի մեծ չափսի դեպքում տուժում է երզրերի հայտնաբերման ճշտությունը, ինչպես նաեւ ավելանում է տեղայնացման սխալը: Օպտիմալ է համարվում 5x5 չափսի միջուկը:

1.2.2 Պատկերի ինտենսիվության գրադիենտի հաշվարկ

Քանի որ եզրերը կարող են ունենալ տարբեր ուղղություններ, Քեննիի ալգորիթմը օգտագործում է 4 ուղղության ֆիլտրեր՝ 0, 45, 90 եւ 135 ակյունների համար: Սոբելի եզրերի հայտնաբերման օպերատորը վերադարձնում է վառության քարտեզում հորիզոնական եւ ուղղահայաց առաջին կարգի ածանցյալները՝ (Gx, Gy), որոնք տեղադրվում են (2) բանաձեւում եզրերի գրադիենտի հաշվարկի համար:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \operatorname{atan2}(G_y, G_x) \qquad (2)$$

1.2.3 Ոչ մաքսիմալ ճնշման կիրառում

Վերջին քայլից հետո գտնված եզրերը հաստ են եւ պարունակում են ավելորդ կետեր: Այդ կետերից ազատվելու նպատակով բոլոր գրադիենտի արժեքները կնվազեցվեն մինչեւ 0, բացի լոկալ մաքսիմումից, որն իրենից ներկայացնում է վառության ամենակտրուկ անցումը՝ փնտրվող եզրը: Ամեն մի փիքսելի համար կատարվում են հետեւյալ քայլերը՝

- Համեմատվում է փիքսելի եզրային ուժգնությունը՝ դրական ու բացասական ուղղության փիքսելների եզրային ուժգնության հետ:
- Եթե տվյալ ուղղությունում դիտարկվող փիքսելի եզրային ուժգնությունն ավելի մեծ է, ապա փիքսելը կմնա եզրի մեջ: Հակառակ դեպքում այն կճնշվի:

1.2.4 Կրկնակի շեմային ֆիլտրում

Մնացած փիքսելները այս փուլում դասակարգվում են բաժանվում են երեք խմբերի՝ ուժեղ, թույլ եւ անպետք: Դա կատարվում է երկու շեմերի սահմանմամբ: Առաջինը կողմում է բարձր շեմ: Բոլոր այն փիքսլենրը, որոնց համար եզրին ուղղահայաց գրադիենտի արժեքը ավելի մեծ է բարձր շեմից դասվում են ուժեքղ եզրերի փիքսելների շարքին: Երկրորդ շեմը ցածր շեմն է: Այն փիքսելները որոնց համապատասխան գրադիենրի արժեքը գտնվում է բարձր եւ ցածր շեմերի միջակայքում համարվում են թույլ եզրերի փիքսելներ: Մնացած փիքսելները ճնշվում են:

1.2.5 Թույլ եզրերի հեռացում

Սովորաբար թույլ եզրերը, որոնք կապակցված չեն ուժեղ եզրերին, իրական եզրեր չեն: Եզրերի կապակցվածությունը հաշվարկվում է Բինար Մեծ Օբյեկնտների մեթոդով, որը հետազոտում է թույլ եզրերի ու դրանց շրջակայքի փիքսելները: Բոլոր թույլ եզրերը, որոնք կապակցված են ուժեղ եզերի հետ պահպանվում են:

Եզրակացություն

Քեննիի եզրերի փնտրման ալգորիթմը եզրերի փնտրման դասական ալգորիթմ է: Քեննիի ալգորիթմի իրականացումը հասանելի է OpenCV բաց գրադարանում: Այն, որպես մուտք ընդունում է պատկերի փիքսելների մոխրագույնի երանգների մատրիցը եւ վերադարձնում է բինար մատրից, որտեղ հայտնաբերված եզրերին պատկանող փիքսելները ունեն 1, մնացածը 0 արժեք:

1.3 K-means կլաստերավորման ալգորիթմ

Տեղեկույթի հաշված քանակի դասերի բաշխումն ըստ որեեւէ հատկանիշի կամ հատկանիշների համախմբի կոչվում է կլաստերավորում: Պատկերների սեգմենտավորումը կլաստերավորման խնդիր է, որտեղ պատկերի փիքսելները ծառայում են, որպես տեղեկույթի կետեր, եւ դրանց անհրաժեշտ է բաշխել ըստ սեգմենտների հաշվի առնելով սեգմենտին պատկանելիության սահմանված չափանիշները:

Ընդհանուր դեպքում ունենք տեղեկույթի n կետեր x_i, i=1...n: Որպես կետեր կարող են ծառայել պատկերի մոխրագույնի երանգներով վառության մատրիցում ամեն մի փիքսելի

արժեքը: Կլաստերների քանակը՝ k-ն, անհրաժեշտ է նախապես սահմանել: Անհրաժեշտ է բոլոր տեղեկույթի կետերը բաշխել k կլաստերներում: Եթե դիտարկենք խնդրի երկրաչափական ներկայացումը, ապա անհրաժեշտ է գտնել μ_i, i=1...k կլատերների կենտրոնների դիրքերն այնպես, որ յուրաքանչյուր կլաստերի կենտրոնից մինչեւ վերջինիս պատկանող կետերը հեռավորության միջին քառակուսայինը լինի մինիմալ: Անալիտիկ տեսքով նշված պայմանը տրվում է (3) արտահայտությամբ, որտեղ c_i-ն i-րդ կլաստերին պատկանող կետերի խումբն է: Կետերի միջեւ հեռավորությունն հաշվարկվում է Էվկլիդյան հեռավորության բանաձեւով $d(x,\mu_i) = ||x - \mu_i||_2$: Սա ոչ բազմանդամային ժամանակի բարդության խնդիր է, այդ իսկ պատճառով ալգորիթմը հույս է ունենում գտնել գլոբալ մինիմումը, սակայն ավարտում է իր աշխատանքը տրված քանակի քայլերի, կամ թույլատրելի մինիմալ սխալին հասնելու դեպքում:

$$\arg\min_{c} \sum_{i=1}^{k} \sum_{x \in c_i} d(x,\mu_i)^2 = \arg\min_{c} \sum_{i=1}^{k} \sum_{x \in c_i} ||x - \mu_i||_2^{\,2} \quad (3)$$

Ալգորիթմը բաղկացած է 4 փուլերից

1. Կլաստերների կենտրոնների պատահական սկզբնարժեքավորում: $\mu_i = rand, i = 1 \dots k$
2. Ամեն մի կետ դասվում է այն կլաստերին, որի կենտրոնն իրեն ամենամոտն է: $c_i = \{j: d(x_j,\mu_i) \le d(x_j,\mu_l), l \ne i, j = 1, \dots, n\}$
3. Յուրաքանչյուր կլաստերի համար տեղադրել միջնակետը՝ կաստերին պատկանող կետերի մեջտեղում: $\mu_i = \frac{1}{|c_i|}\sum_{j \in c_i} x_j, \forall i$
4. Շարունակել 2,3 կետերն այնքան ժամանակ մինչեւ, կամ կլաստերների կենտրոնների տեղաշարժը նախորդ իտերացիայի համեմատ լինի ավելի փոքր նախապես տրված շեմից, կամ էլ իտերացիաների քանակը հասնի նախապես տրված թվին:

1.4 L*a*b* գունային տարածություն

1.5 Սուպերփիքսելային սեգմենտավորում

Օբյեկտների տարանջատման խնդրի լուծումը ենթադրում է

Պատկերների սեգմենտավորման դասական ալգորիթմներում, որպես մշակման միավոր առավելապես ընդունվում է մեկ փիքսելը իր հատկանիշներով՝ գույն, դիրք եւ այլն: Որպես

1.6 SLIC սուպերփիքսելային սեգմենտավորման ալգորիթմը

1.1.1 Sound Engine և դաշնամուր

Windows Phone օպերացիոն համակարգը տրամադրում է ձայնային ֆայլերի հետ աշխատելու համար դասեր, որոնք սահմանված են Microsoft.Xna.Framework.Audio անունների տարածությունում: Այստեղ կան երկու դասեր SoundEffect և SoundEffectInstance: Առաջինը թույլ է տալիս աուդիո ֆայլի Stream-ի կամ բայթ զանգվածից ստեղծել փոփոխական և տալ ֆայլի հաճախականությունն ու աուդիո կանալների քանակը: Այն ունի նաև նվագարկելու ֆունկցիա, սակայն չունի դադարի ֆունկցիոնալություն այդ իսկ պատճառով սահմանափակ է իր հնարավորություններով: Դադարի, վերսկսման և ավարտի ֆունկցիաներ է տրամադրում երկրորդ՝ SoundEffectInstance դասը, որի փոփոխական կարելի է ստանալ SoundEffect օբյեկտի վրա կանչելով CreateInstance մեթոդը:

Դաշնամուրի հավելված ստեղծելու համար անհրաժեշտ է նախ ընտրել քանի նոտաներ են ներառվելու և դրանց համապատասխան ձայների սեմփլները կապել ինտերֆեյսի ստեղների սեղմման հետ: Տվյալ աշխանանքում պարզության համար կա նոտաների մեկ օկտավա, այսինքն 11 ստեղն:

1.2 Windows Phone 8 օպերացիոն համակարգ

1.2.1 Ֆայլերը և դրանց նկատմամբ թույլատվությունները

Տվյալ աշխատանքի շրջանակներում մշակվում է Windows Phone 8 օպերացիոն համակարգի համար հավելված: Ինչպես արդեն նշվեց այս համակարգը մշակվել է Microsoft ընկերության կողմից և հանդիսանում է 3-րդ օպերացիոն համակարգը աշխարհում, ըստ օգտագործողների քանակի: Այն տրամադրում է գործիքամիջոցների հավաքածու, որը կոչվում է Windows Phone SDK, որոնց միջոցով իրականացվում է հավելվածների մշակումը տվյալ պլատֆորմի վրա: Գործիքամիջոցները սահմանում են այն գործողությունները, որոնք թույլատրվում է կատարել հավելվածի մեջ: Այն ունի մի շարք սահմանափակումների կապված ֆայլերին մուտք ստանալու հետ: Դիտարկենք թե ինչ հիմնական տեսակի և նշանակության ֆայլերի հետ կարելի է աշխատել:

- Ֆոտո ֆայլեր: Իրենցից ներկայացնում են JPG, BMP, PNG և այլ ֆորմատի պատկերներ, որոնք պահվում են Windows Phone համակարգի Նկարներ

(Pictures) պանակում: Նրանք պահպանվում են ֆոտո ալբոմների տեսքով: Այդ ալբոմների տեսակավորված են ըստ նշանակության:

- Camera Roll – Այստեղ են պահվում տեսախցիկի միջոցով նկարված լուսանկարները: Նրանք կախված տեսախցիկի պարամետրերից կարող են տատանվել 800 կիլոբայթից մինչև 5 մեգաբայթ:
- Saved Pictures – Այստեղ են պահվում արտաքին աղբյուրներից ներբեռնված նկարները: Որպես այդպիսի աղբյուր կարող են հանդես գալ ինտերնետային կայքերը, հավելվածները կամ միացված համակարգչից պատճենված ֆայլերը:
- Այլ ալբոմներ, որոնք կարող են ստեղծվել թե օգտագործողի կողմից և թե կցված սոցիալական կայքերից: Օրինակ՝ եթե ունեք Ֆեյսբուք սոցիալական կայքի հեռախոսին կցած փրոֆայլ, ապա ձեր Ֆեյսբուքյան ալբոմները նույնպես պատճենված կլինեն Նկարներ բաժնում:

Windows Phone 8 օպերացիոն համակարգում նկարների հետ թույլատրվում է կատարել կարդացման և գրանցման գործողություններ: Ըստ այդմ կարելի է պատճենահանել օգտագործողի բոլոր նկարները և պահպանել թաքուն: Սակայն չի թույլատրվում ջնջել այդ ֆայլերը նախնական պահուստից: Սա նշանակում է, որ օգտագործողը ինքն է պատասխանատու թաքստոց պատճենահանելուց հետո ջնջել այդ նկարները Նկարներ բաժնից:

- Կոնտակներ: Դա մարդկանց մասին տվյալներն են, որոնք պահվում են կցված, որևէ սոցիալական կայքի փրոֆայլին (Օրինակ Google account, Microsoft account): Կոնտակները պահվում են Մարդիկ (People) բաժնում: Windows Phone 8 համակարգում կոնտակտը պարունակում է հետևյալ տեղեկատվությունը
  - Փրոֆայլներ (Accounts) – այն սոցիալական կայքերի տեսակներն ու տվյալները, որոնց կցված է տվյալ կոնտակտը
  - Հասցեներ (Addresses) – կոնտակտի բնակության հասցեն է, որը պարունակում է երկու հասցեական տողեր, շենքի համար, քաղաք, աշխարհագրական ռեգիոն, շինության հարկ, փոստային կոդ և մարզ:
  - Երեխաներ (Children) – կոնտակտի երեխաների անուն ազգանունների ցուցակ

- Ընկերություններ (Companies) – կոնտակտի աշխատավայրերի մասին տեղեկատվություն, որը ներառում է ընկերության անվանումը, կոնտակտի պաշտոնը, գրասենյակի հասցեն և ընկերության Ճապոներեն լեզվով անվանումը(անհրաժեշտության դեպքում)
- Լրիվ անունը (CompleteName) – սա կոնտակտի անունն է, որը ներառում է՝ անուն, ազգանուն, միջին անուն, մականուն, նախդիր, կոչում և անունն ու ազգանունը Ճապոներենով
- Էլեկտրոնային փոստի հասցեները (Email Addresses) – օգտագործողի էլ: փոստի հասցեները
- Գրառումներ (Notes) – տվյալ կոնտակտի վերաբերյալ տողային գրառումներ
- Հեռախոսի համարները (PhoneNumbers) – կոնտակտի հեռախոսի համարները և տեսակները (բջջային, աշխատանքային, տնային և այլն)
- Ձույգ (SignificantOther) – կոնտակտի ընկերոջ(ընկերուհու) անունն ու ազգանունը
- Վեբ կայքեր (Websites) – կոնտակտի վեբ կայքերը

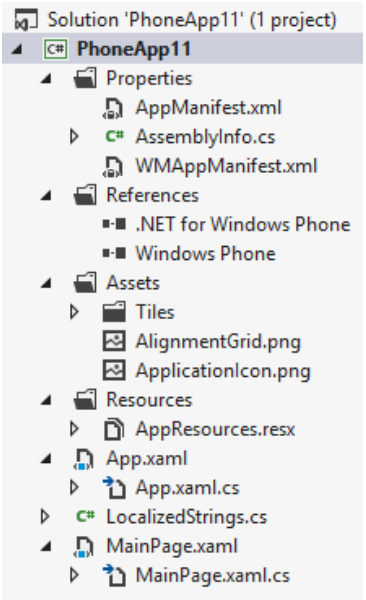
Կոնտակների հետ թույլատրվում է կատարել կարդալու և գրելու գործողություն: Ըստ այդմ կարելի է պաճենահանել օգտագործողի բոլոր կոնտակները և պահպանել թաքուն: Սակայն չի թույլատրվում դրանց ջնջել նախնական պահուստից: Սա նշանակում է, որ օգտագործողը ինքն է պատասխանատու թաքստոց պատճենահանելուց հետո ջնջել այդ կոնտակները Մարդիկ բաժնից:

- Աուդիո և վիդեո ֆայլեր: Իրենցից ներկայացնում են բազմազան աուդիո/վիդեր ֆորմատների(MP3, Wav, AAC/MP4, WMV և այլն) ֆայլեր: Նրանց ցավոք կարելի է դիմել միայն երաժշտական Music ծրագրի միջոցով: Ծրագրային գործիքաշարը թույլատրում է այդ ֆայլերի նվագարկումը, սակայն միայն տրամադրվաշ ինտերֆեյսի՝ լաունչերի միջոցով, որը արգելում է այդ ֆայլերի ուղիղ կարդացումը հոսքի միջոցով, հետևաբար և պատճենումը: Սա արված է հեղինակային իրավունքերի պաշտպանության նկատառումներով:

- Դոկումենտներ: Ներառում են Microsoft Office ծրագրային համալիրի ֆորմատների ինչպես նաև PDF ֆորմատի ֆայլերը: Այս ֆայլերը, սակայն կարելի է միայն բացել հատուկ տրամադրված ծրագրերի Adobe Reader-ի և Microsoft Office-ի միջոցով և գործիքաշարինց նրանք անհասանելի են:

1.2.2 Windows Phone 8 հավելվածի նախագիծ

Windows Phone 8 օպերացիոն համակարգի ներքո որպես հավելվածների պատրաստման համար նախատեսված Ծրագրավորման Ինտեգրացված Միջավայր (IDE) է ծառայում Microsoft Visual Studio ծրագրի 2011 և ավելի նոր վերսիաները: Ծրագրավորողի գործը պարզեցնելու համար այնտեղ SDK-ի տեղադրման ժամանակ նաև տեղադրվում են նախագծի շաբլոններ, որոնք իրենցից ներկայացնում է ծրագրի դատարկ կմախք՝ բաղկացած մինիմալ անհրաժեշտ ֆայլերից, կամ ավելի բարդ համակարգ, որի մեջ արդեն իսկ կան Windows Phone հավելվածների ամենատարածված ինտերֆեյսների իրականացումները: Պարզագույն դատարկ շաբլոնը Windows Phone App շաբլոնն է, որը ստեղծելիս տալիս ենք նախագծի անունը և արդյունքում ստեղծվում են մի շարք ֆայլեր, որոնք կարող ենք տեսնել Solution Explorer պատուհանում (նկ. 2.1):



Նկ. 2.1 Solution Explorer պատուհանը Windows Phone App դատարկ շաբլոնով նախագիծ պատրաստելիս

Ստեղծվել է Solution կոնտեյները, որը իրենից ներկայացնում է նախագծին վերաբերվող բոլոր ֆայլերի հիմնային կոնտեյներ: Նրա մեջ ստեղծված է PhoneApp1 անունով

նախագիծ: Նախագիծը դա այն մի կոնտեյներ է, որի վրա կարելի է կատարել Run և Debug գործողությունները: Այն կազմված է հետևյալ բաժիններից.

- Հակություններ (Properties) – Պարունակում են նախագծի փաստաթղթաբանական և թույլատվությունների մասին տեղեկատվություն: Դրանց շարքին են պատկանում նախագծի անունը, ընկերության անունը, հեղինակային իրավունքը, ծրագրի վերսիան, թույլատվությունը կոնտակներին, տեսախցիկին և այլմ
- Հղումներ (References) – Պարունակում է ծրագրի հղումները արտաքին դինամիկ շաղկապման գրադարաններին և վեբ սերվիսներին: Օրինակ կարող է պարունակել հղում ձայնային գրադարանի վրա, դաշնամուրի հավելված ստեղծելիս ձայնային հնարավորությունները ախահովելու համար:
- Ակտիվներ (Assets) – Պարունակում է ծրագրի կողմից օգտագործվող ռեսուրսային ֆայլերը, ինչպիսիք են պատկերները, ձայները և այլն
- Ռեսուրսներ (Resources) և Լոկալիզացված տողեր (LocalizedStrings)– Պարոնակում է .resx ընդլայնման ֆայլեր, բազմաթիվ լեզուներ ապահովող հավելվածների լեզուների ռեսուրները պահելու համար:
- App.xaml և App.cs – ծրագրի մուտային կետը գտնվում է այս ֆայլում: Այն տրամադում է իրադարձությունների բռնոտիչներ ծրագրի սկզբի, ավարտի, սառեցման և վերաակտիվացման ժամկանակ, որպեսզի ծրագրավորողը կարողանա կատարել վիճակների պահպանում և բեռնում այդ իրադարձություններից առաջ:
- MainPage.xaml և MainPage.cs – պարունակում է ծրագրի առաջին էջը, որը երևում է, երբ բացում ենք ծրագրիը: Ընդհանուր առմամբ հավելվածը կարող է բաղկացած լինել մեկ և ավելի էջերից: Xaml ֆայլում XAML լեզվով նկարագրվում է էջի արտաքին ինտերֆեյսը, իսկ cs ֆայլում գրվում է այդ էջի ֆունկցիոնալությունը ապահովող կոդը C# ծրագրավորման լեզվով:

1.2.3 MVVM փաթերնը

Windows Phone Silverlight հենքի վրա պատրաստվող ծրագրերի, ինչպես և .NET-ի այլ XAML ինտերֆեյս ունեցող ծրագրերի համար առաջարկվում է օգտագործել MVVM

(Model View View Model) ծրագրավորման փաթերնը: Այն նախատեսված է ընդլայնվող և կոմպակտ կոդ ունեցող հավելվածներ մշակելու համար:

MVVM փաթերնում իրարից ֆիզիկապես բաժանվում են ինտերֆեյսային և տրամաբանության կոդրը: Այդպես ինտերֆեյսը դառնում է XAML լեզվով նկարագրված ֆայլը, որը կոչվում է View: Նույն ինտերֆեյսը կարող է օգտագործվել տարբեր տեղերում տարբեր տեղեկատվություն ցուցադրելու նպատակով: Ինտերֆեյսին տվյալներով է մատակարարում Model-ը, որը պարունակում է ծրագրի այդ էջի տվյալներն ու տրամաբանությունը: Մոդելը դա դաս է, որը պարունակում է View –ին անհրաշեշտ բոլոր դաշտերը: Մոդելի օբյեկտների ստեղծմամբ զբաղվում է ViewModel դասը, որը մի կողմից կապվում է մոդելի հետ պարունակելով վերջինիս օբյեկտներ, մյուս կողմից View-ի հետ: Վերջին կապը իրականացվում է Binding կոչվող տեխնոլոգիայի միջոցով: View –ի ամեն ատրիբուտին, որը պետք է ինֆորմացիա ստանա ViewModel-ից, վերագրվում է Binding Model-ի այդ դաշտին, որը այդ ատրիբուտը պետք է պարունակի: Դա կարող է ունենալ հետևյալ տեսքը՝

< TextBlock Text={Binding TextValue}/>

Այս գրելաձևը XAML լեզվով հայտարարում է տեքստային դաշտ և նշում, որ որպես վերջինիս Text ատրիբուտի արժեք է ծառայում Model-ի TextValue հատկությունը: Մոդել դասը պետք է իրականացնի INotifyPropertyChanged ինտերֆեյսը, որպեսզի նրա ցանկացած դաշտի փոփոխություն անմիջապես արտացոլվի արտաքին ինտերֆեյսում իրեն Bind եղած էլեմենտերի միջոցով: ViewModel-ը իր հերթին կարող է պարունակել ոչ թե մոդելի մեկ օբյեկտ, այլ օբյեկտների զանգված: Այս դեպքում զանգվածը պետք է իրականացնի INotifyCollectionChanged ինտերֆեյսը, որպեսզի տեղեկացնի արտաքին ինտերֆեյսին, օբյեկտների ավելացման կամ հեռացման մասին: Այդպիսի զանգված է հանդիսանում, օրինակ, ObservableCollection դասի օբյեկտը: View-ն ViewModel-ի կապելու համար ընդամենը անհրաժեշտ է View.DataContext = ViewModel: Դատա կոնտեկստը այն օբյեկտն `է, որի մեջ Binding մեխանիզմը փնտրելու է TextValue անունով հատկություն և գտնելու դեպքում արժեքը վերագրելու TextBlock.Text ատրիբուտին: Եթե ընթացքում այդ արժեքը փոփոխվի ViewModel-ի միջոցով, ապա ViewModel-ը

կտեղեկանա այդ մասին PropertyChanged իրադարձության միջոցով և կփոխանցի այդ ինֆորմացիան View-ին, որը կթարմացվի՝ Text-ին վերագրելով նոր արժեքը:

1.2.4 Ֆայլային համակարգի API

Windows Phone SDK-ը տրամադրում է ֆայլերի հետ աշխատելու 4 մեթոդներ, որոնք միմյանցից տարբերվում են ֆայլերի տեղակայման կոնտեյներով: Տվյալ աշխատանքում օգտագործվում են Isolated Storage տարածքը, որը ծրագրին տրամադրված սկավառակային հիշողության տարածք է, որից կարդալու և գրելու հնարավորություն ունի միայն տվյալ հավելվածը: Ֆայլը պահվում է հոսանքի (Stream) տեսքով: Իզոլացված հիշողությունում կարելի է ստեղծել/ջնջել/փնտրել/վերանվանել պանակներ և ֆայլեր: Դա ապահովում են API-ի ֆունկցիաները:

Եթե ծրագրում տեղեկատվությունը, որը անհրաժեշտ է պահպանել ներկայացվում է օբյեկտների տեսքով, ապա անհրաժեշտություն է առաջանում այդ օբյեկները վերածել հոսանքի և այդ հոսանքի մեջ պետք է պահպանվի նաև այն դասի և դասի դաշտերի մասին ինֆորմացիան, որի օբյեկտը պետք է պահել: Սա անհրաժեշտ է, որպեսզի օբյեկը հնարավոր լինի ինիցիալիզացնել՝ այն կարդալով հիշողությունից հոսանքի տեսքով: Նկարագրված պրոցեսսները կոչվում են սերիալիզացիա և դեսերիալիզացիա: Սերիալիզացիայի տարածված տարբերակ է օբյեկտները JSON (Javascript Serialization) ֆորատի կոնվերտացնելը: Այդ ֆորմատը նկարագրում է դասի դաշտերը և դրանց արժեքները առանց կորուստների: Սակայն այս ֆորմատի սերիալիզացնելու համար անհրաժեշտ է, որպեսզի դասում պարունակվեն միայն սերալիզացվող տիպի անդամներ: Դա պարզ ներդրված տիպերն են, օրինակ՝ int, string: Սա նշանակում է, որ օրինակ BitmapImage նկար պարունակող դասի օբյեկտը հնարավոր չէ սերալիզացնել: Այս խնդրից ելնելով սույն աշխատությունում մշակվել է գործելակարգ, որով հնարավոր է հոսանքի տեսքով պահել օբյեկ, որը պարունակում է նկար և հաջողությամբ այն դեսերիալիզացնել:

Սերիալիզացիայի քայլերը հետևյալն են.

- DataContractJsonSerializer դասի միջողով սերալիզացվում են դասի այն բոլոր դաշտերը, որոնք ունեն պարզ տիպ:

- Հաշվվում է սերիալիզացված հոսանքի երկարությունը և պահվում int տիպի փոփոխականի մեջ
- Այդ թիվը վերածվում է 4 երկարությամբ բայթ զանգվածի, ապա հոսանքի և գրանցվում պահվող ֆայլի սկզբում
- Սերիալիզացված ֆայլը ավելացվում է պահվող ֆայլի վերջում
- Նկարի հոսանքը գրանցվում է ֆայլի վերջում

Դեսերիալիզացիայի քայլերը հետևյալն են.

- Կարդացվում է առաջին 4 բայթը և կոնվերտացվում Int32 տիպի Length փոփոխականում
- Կարդացվում է հաջորդ Length քանակությամբ բայթերը և դեսերիալիզացվում DataContractJsonDeserializer դասի օբյեկտի միջոցով
- Կարդացվում է մնացած բայթերը և վերագրվում դեսերիալիզացված օբյեկտի նկարին

Գլուխ 2: Խնդրի դրվածքը և կատարված աշխատանքի ալգորիթմի ուսումնասիրությունը

2.1 Խնդրի դրվածքը

Տեղեկատվական տեխնոլոգիաների ոլորտում 20-րդ դարի սրընթաց զարգացումը հանգեցրեց տնային համակարգիչների տարածմանը և այսօր այդ երևույթը սովորական է բոլորիս համար: Եթե 20-րդ դարի ՏՀՏ ոլորտի ամենատարածված պրոդուկտը համարենք տնային համակարգիչը, ապա 21-րդ դարում այն իր տեղն է զիջում սմարթֆոններին և պլանշետային համակարգիչներին: Վերջիններս իրենց ֆունկցիոնալությամբ այսօր չեն զիջում տնային ստատիկ համակարգիչներին և իրենց հերթին ավելացնում են օգտատերի համար մի շատ կարևոր արժեք՝ դյուրակիրություն: Դա նշանակում է ունենալ համակարգիչ ցանկացած պահի ցանկացած վայրում: Օգտատերերի կողմից սմարթֆոնների կիրառման հիմնական եկրու առարկաներն են՝

- Սմարթֆոնը, որպես զանգեր կատարելու և կարճհաղորդագրություններով շփվելու միջոց (այսինքն սմարթֆոնի այն գործառույթները, որոնք իրականացնում են հասարակ բջջային հեռաղոսները)
- Սմարթֆոնը, որպես միջոց օգտվելու տվյալ հենքի համար պատրաստված հավելվածներից (application)
- Սմարթֆոնը, անձնական տեղեկույթի պահպանման, օգտագործման և մշակման միջոց

Վերջին կետը ներառում է նկարների, կոնտակների, տեսահոլովակների, աուդիոֆայլերի, տեքստային դոկումենտների և այլ մուլտիմեդիոն, փաստաթղթային ֆայլերի հետ աշխատանքը:

Դյուրակիր սարքերի ի հայտ գալը բազմաթիվ հարմարություններից բացի առաջ բերեց մի կարևոր խնդիր օգտագործողի համար: Խնդիրը կայանում է անձնական տվյալները այլ անձանցից անվտանգ և շատ դեպքերում նույնիսկ թաքուն պահելու անհրաժեշտության: Օգտագործողը սմարթֆոնը ընկերոջ խնդրանքով կամ պատահաբար հայտնվելով այլ կողմնակի անձի մոտ հնարավարություն է ընձեռում վերջինիս դիտարկել, փոփոխել և նույնիսկ ջնջել տվյալներ, բավարար անվտանգության միջոցառումների բացակայության պարագայում: Առանձին ուշադրության են արժանի այն տվյլաները, դեպի որոնց մուտքի սահմանափակումը բավարար չէ օգտատիրոջ համար և նրան անհրաժեշտը գաղտնի տվյլաները թաքուն պահել, որպեսզի կողմնակի անձանց մոտ կասկած չառաջանա այդ տվայլենրի գոյութան մասին:

Տվյալ աշխատանքը ծառայում է վերընշված խնդրի լուծմանը: Այն իրենից ներկայացնում է մոբայլ հավելված, որը ունենալով երկու տարբեր կեղծ և իրական ինտերֆեյսներ չարոնագրված մարդկանց ցույց է տալիս կեղծ ինտերֆեյս, իսկ արտոնագրվածներին հատուկ դիմակի միջոցով թույլ է տալիս կեղծ ինտերֆեյսից անցում կատարել իրական ինտերֆեյս, որտեղ օգտատերը կարող է պահել և դիտարկել իր անձնական մուլտիմեդիոն ֆայլերն ու փաստաթղթերը:

Ծրագրին գրվել է Windows Phone 8 պլատֆորմի համար ինչը ունի հետևյալ հիմնավորումները՝

- Այսօրվա դրությանբ սմարթֆոնների համաշխարհային շուկայում դոմինանտ են 3 մոբայլ օպերացիոն համակարգեր՝

- iOS, որը պատկանում է Apple կորպորացիային
- Android, որի սեփականատերն է Google կորպորացիան
- Windows Phone, որը պատկանում է Microsoft կորպորացիային

- Andoid օպերացիոն համակարգը հիմնված է Unix օպերացիոն համակարգի վրա և իր բնույթով նշված 3 օպերացիոն համակարգերից ամենաբացն է: Այս հենքի վրա ստեղծվելիս ծրագրավորողներին թույլ է տրված մուտք գործել ֆայլային համակարգ այնտեղ կատարելով ցանկալի գործողություններ: Սա պատճառ է հանդիսանում մի շարք հավելվածների գոյությանը (օրինակ՝ ES File Explorer), որոնց ֆունկղիոնալությունը ներառում է ֆայլերի թաքուն դարձնելը: Սա հիմնավորում է առաջադրված խնդրի լուծումների առկայությունը Andoid հենքում և հանում այն դիտարկման ցուցակից
- Մնում են iOS և Windows Phone օպերացիոն համակարգերը, որոնց համար հավելվածների ստեղծման հիմնական ծրագրավորման լեզուներն են համապատասխանաբար Objective C և C# լեզուները: Հեղինակի կողմից C# լեզվի և Windows Phone SDK ծրագրավորման կիրառական գործիքների իմացությունը հիմք հանդիսացավ հենց Windows Phone համակարգի ընտրման համար
- Windows Phone համակարգի հավելվածների շուկայում կատարված հետազոտությունների արդյունքում չի հայտնաբերվել առաջադրված խնդրի պահանջներին բավարարող լուծում

Ծրագրի ստեղծման ընթացքում կիրառվել են Windows Phone SDK (Software Development Kit/Ծրագրային ապահովման ստեղծման գործիքաշար) գրադարաններ, ինչպես նաև ստեղծվել են սեփականանները:

Ծրագրիավոման միջավարն է Microsoft Visual Studio 2013, որը լինելով Microsoft ընկերության պրոդուկտ մաքսիմալ կերպով հարմարեցված է C# լեզվի հետ աճխատանքի համար և հանդիսանում է միակ օֆիցիալ միջավայրը, որտեղ կարելի է կիրառել Windows Phone SDK գործիքաշարը:

Գրականույթան ուսումնասիրության արդյունքում եզրակացնու ենք, որ մեր առջև հնարավոր է դնել Windows Phone 8 օպերացիոն համակարգում ֆոտո ֆայլերի և կոնտակների թաքցնումը հատում հավելվածի միջոցով: Թաքցնումը ենթադրում է, որ ծրագրի մեջ պատճենելուց հետո, օգտագործողը ինքն է պարտավոր ջնջել այդ ֆայլերը նախնական աղբյուրներից:

2.2 Թաքցման դիմակ

Խնդրի դրվածքում արդեն, նշվեց, որ անձնական ինֆորմացիան պետք է թաքցվի հավելվածի միջոցով, որը ունի 2 ինտերֆեյսներ՝ կեղծ և իրական: Կեղծ ինտերֆեյսի պարամետրերի կոնկրետ վիճակը կանվանենք թաքցման դիմակ: Դիմակը նկարագրելուց առաջ նախ ծանոթանանք կեղծ ինտերֆեյսի կառուցվածքին:

Որպես կեղծ ինտերֆեյս պետք է ծառայի այնպիսի մի կոնստրուկցիա, որը տարածված է մոբայլ համակարգերի շուկաներում, ունի լայն շրջանառություն ցանկցած տարիքային կատեգորիաների համար: Կեղծ ինտերֆեյսը ցանկացած ոչ իրավասու անձի համար պետք է ներկայանա, որպես լիարժեք հավելված, որը ունի որոշակի ֆունկցիոնալություն և իր առկայությամբ կասկած չի առաջանցում: Պետք է հնարավոր չլինի ոչ իրավասու անձի կողմից իրական ինտերֆեյսի գոյության մասին ենթադրություն անելը:

Վերը նշված պայմանները հաշվի առնելով, որպես կեղծ ինտերֆեյս է ընտրվել դաշնամուր նվագելու համար նախատեսված պարզ հավելվածը: Այն ունի շատ պարզ կառուցվածք պարունակում է դաշնամուրի մեկ օկտավա, այսինքն 11 ստեղներ (նկ.): Այդպիսի հավելվածները բազում են շուկայում և ունեն մեծ պահանջարկ, թե մեծերի և թե փոքրերի մոտ, թե երաժշտությամբ զբաղվողների և թե ուղղակի երաժշտասերների մոտ: Ծրագրի անունն է Simple Piano, այսինքն պարզ դաշնամուր: Ցանկացած օգտագործող անկախ իրավասություններից կարող է բացել և օգտագործել ծրագրի կեղծ ինտերֆեյսը, առանց անգամ ենթադրելու իրական ինտերֆեյսի գոյության մասին:

Իսկ ի՞նչ պարամետրերի վիճակ է կարելի սահմանել այս ինտերֆեյսի համար, որը հենց կլինի դիմակը: Դիմակը իրենից ներկայացնում է, որոշակի հաջորդականությամբ նվագված ստեղների հաջորդականություն: Ճիշտ դիմակ նվագելու դեպքում ծրագիրը բացում է իր իրական թաքուն ինտերֆեյսը, որի մասին կխոսվի ավելի ուշ: Մնում է, որոշել ինչ ալգորիթմով է կառուցվում դիմակը:

Դիմակի կառուցումից առաջ նախ եկեք սահմանենք այն պահանջները, որոնք ներկայացնում ենք դիմակին:

- Դիմակը պետք է կազմվի օգտագործողի կողմից տրամադրվող գաղտնի բանալու կողմից: Այս պայմանը ապահովում է գաղտնաբանության հիմնական դրույթներից մեկը, որը սահմանում է, որ գաղտնագրված կամ թաքնագրված տեղեկատվության անվտանգությունը պետք է միմիայն կախված լինի գաղտնի

բանալիից և ոչ թե ալգորիթմից: Այս պայմանը տեղի ունի քանի որ միշտ պետք է ենթադրել, որ հակառակորդին հասանելի է դիմակի ստացման ալգորիթմը: Պայմանի բացատրությունը շատ պարզ է: Ենթադրենք համակարգը կոտրված է, այսինքն հակառակորդը գտել է գործնական միջոց դիմակը, որոշելու համար: Այդպիսի իրավիճակում, եթե համակարգի կայունությունը հիմնված լինի դիմակի ստացման ալգորիթմի վրա, անհրաժեշտություն կառաջանա սուղ ժամկետներում նոր ալգորիթմ մշակել ինչը դժվար է և կապված է մեծ ծախսերի հետ: Այդ իսկ պաճառով համակարգի կայունությունը դրվում է բանալիի վրա և կոտրման դեպքում ընդամնեևը անհրաժեշտ է փոխել բանալին, ինչը պարզ գործընթաց է:

- Դիմակը պետք է չլինի ստատիկ: Այս պայմանը կապված է բանալիների կյանքի աշխատանքի կարևոր ցիկլերից մեկի՝ բանալիների փոփոխման հետ: Այս պայմանը ենթադրում է, որ անվտանգության նկատառումներից անհրաժեշտ է ժամանակ առ ժամանակ փոփոխել դիմակը: Այս պայմանի հիմնավորումը կայանում է նրանում, որ նույն երաժշտական ստեղների հաջորդականության անընդհատ ամեն անգամ նվագելը կարող է կասկած առաջանցնել կամ ուղղակի դրդել ոչ իրավասու անձին ով ականջ է դրել ծրագրի աշխատանքին, նույնպես փորձել նվագել այդ հաջորդականությունը: Այդ դեպքում համակարգը կոտրված կլինի: Այս նկատառումներից ելնելով ցանկալի կլինի, որ դիմակը փոփոխական լինի և փոփոխությունը կախված բազմաթիվ պարամետրերից: Օգտագործողը պետք է ցանկացած պահի կարենա փոփոխել դիմակի իրեն հայտնի գաղնի բանալին:
- Դիմակը պետք է լինի բավական երկար, որպեսզի հատարկման եղանակով այն հնարավոր չլինի գտնել և պետք է լինի բավական կարճ, որպեսզի օգտագործողին հարմար լինի այն հիշել և օգտագործել:
- Դիմակը պետք է լինի թվերի հաջորդականություն, որը օգտագործողը պետք է նվագի դաշնամուրի վրա:

Թաքցման դիմակի համար ուսումնասիրությունների արդյունքում առաջադրվել են հետևյալ տարբերակները:

2.2.1 Դիմակ հիմնված ժամանակի վրա

Դիմակի ստեղծման պարտադիր պայմաններից, ինչպես նշվեց ավելի վաղ, հանդիսանում է նրա փոփոխական բնույթը: Փոփոխականությունը պետք է հիմնվի օգտագործողի գաղտնի բանալիի և դիակի ալգորիթմի վրա, որը կախված փոփոխական պարամետրերից կձևափոխի գաղտնի բանալին ստանալով վերջնական դիմակը: Այդ ամենով հանդերձ օգտագործողի և ծրագրի համար պետք է միշտ միարժեքորեն, որոշված լինի մեկ ճիշտ դիմակը: Ժամանակի վրա հիմնված դիմակի դեպքում փոփոխականությունը կապվում է այն հասարակ փաստի հետ, որ ժամանակը ժամացույցի վրա անընդհատ փոփոխվում է: Այդ տրամաբանությունից ելնելով օգտագործողի բանալիի և տվյալ պահին ժամացույցի ցուցադրված ժամի հետ կատարելով, որևէ պարզ գործողություն կարելի է ստանալ դիմակը:

Դիտարկենք հետևյալ ալգորիթմը: Օգտագործողը պահում է չորս միշանոց գաղնտաբառ իր մտքում և միշտ հիշում այն: Այն փաստը, որ կրեդիտ քարտերի Փին կոդերի երկարությունը 4-5 նիշ է նշանակում է, որ ցանկացած մարդու համար չորս թիվ հիշելը դժվարություն չի ներկայացնի: Այդ չորս նիշերը ցանկության դեպքում կարելի է փոփոխել (եթե, օրինակ, կասկածում եք, որ ձեր մոտ տեղի է ունեցել համակարգի կոտրում): Ցանկացած պահի դիմակը որոշվում է գաղտնաբառի ամեն նիշին ըստ մոդուլ տասի ժամացույցի համապատասխան արժեքը գումարելով:

$A_1A_2A_3A_4$ – չորս նիշանոց գաղտնաբառ, յուրաքանչյուր նիշը 0-9 սահմաններում

H_1H_2:M_1M_2 - ժամացույցի ցույց տված ժամը

$D_1D_2D_3D_4$ - դիմակ

Դիմակի ստացման բանաձևը կորոշվի նշված հավասարություններով:

$$D_1 = (A_1 * H_1) mod 10$$
$$D_2 = (A_2 * H_2) mod 10$$
$$D_3 = (A_3 * H_3) mod 10$$
$$D_4 = (A_4 * H_4) mod 10$$

Այս մոտեցման հիմնական առավելությունը կայանում է պարզության և հեշտ հիշվող լինելու մեջ: Բացի այդ այն ավելի քան կայուն է գրոհների նկատմամբ : Եթե ենթադրենք, որ կատարվել է հարձակում հատարկման եղականով, ապա հնարավոր տարբերակների քանակը կլինի 10000: Եթե համարենք, որ ամեն տարբերակի

փորձարկումը տևում է միջինում 3 վարկյան, ապա անհրաժեշտ կլինի T = 30000 վարկյան =500 րոպե: Քանի, որ գաղտնաբառը փոփոխվում է ամեն րոպե, հետևում է, որ հատարկման եղանակով դիմակը կռահեռ գործնականում անհնար է: Այս մոտեցման թույլ կողմը կայանում է նրանում, որ եթե հակառակորդը իմանա դիմակի ստացման ալգորիթմը և ֆիքսի նվագված երաժշտությունը ու ժամանակը, ապա նա կարող է տիրանալ գաղտնի բանալիին: Այս գրոհին կարելի է հակազդել, օրինակ օգտագործելով ականջակալներ, որպեսզի հակառակորդը չլսի ինչ եք դուք նվագում: Կամ կասկածի դեպքում պետք է փոփոխել գաղտնի բանալին:

2.2.2 Դիմակ հիմնված կոորդինատների վրա

Ժամանկի վրա հիմնված համակարգում գաղտնաբառի հավաստիությունը հիմնվում էր այն փաստի վրա, որ հեռախոսի ցուցադրած ժամը և ծրագրավորողի կողմից գործիքաշարի միջոցով ստացված ժամը համընկնում են: Ընդհանրապես դինամիկ փոփոխվող դիմակը կարող է հիմնվել միայն այնպիսի պարամետրերի վրա, որոնց արժեքը օգտագործողի և ծրագրի համար միարժեքորեն և նույնաբար որոշված են: Այդպիսի պարամետրեր կարող են ծառայել GPS համակարգի միջոցով ստացվող լայնական և երկայնական կոորդինատների զույգը: Այդ կոորդինատները իրենցից ներկայացնում են ֆիկսված ստորակետով տվեր, որոնց կոտորակային մասը ունի վեց նիշի ճշտություն: Հետևաբար, նրանցից յուրաքանչյուրը առնվազն 7 նիշով է որոշվում: Կարելի է կատարել հատուկ գործողություններ այդ նիշերի և օգտագործողի գաղտնի բանալիի հետ և ստանալ դիմակը: Այսինքն կստանանք դիմակ, որը ճիշտ է միայն ձեր դիրքում, և եթե ինչ-որ մեկը վերցնի այն և փորձի կրկնել գտնելով ձեզանից մի քանի մետր հեռավորության վրա, ապա այդ դիմակը սխալ կլինի: Սա լավ մոտեցում է, սակայն ունի իր թերությունները: Քանի որ գործ ունենք մեծ թվերի հետ առաջանում է մտովի հաշվարկ կատարելու խնդիր, որը չէր առաջանում ժամացույցի օգտագործելու դեպքում: Այդ խնդրի պարզագույն լուծում կարող է լինել մի ուրիշ օգնական հավելվածի ստեղծումը, որը իբրև թե գեներացնում է պատահական հաջողակ թիվ: Այն նույնպես կասկած չի առաջացնում և կարող է հաշվել կոորդինատներից ստանալ չորս նիշանոց թիվ, որը արդեն կարելի է նիշ առ նիշ, ըստ մոդուլ տասի գումարել, ինչպես արվում էր ժամանակի վրա հիմնված դիմակում: Այս մոտեցման թերությունը կայանում է երկրորդ հավելվածի անհրաժեշտության մեջ և կարող է կասկած առաջացնել

դաշնամուրի ծրագրիը ամեն անգամ օգտագործելիս օգտագործողի կողմից ընթացիկ ծրագրի փոփոխումը դեպի հաջողակ թվի ծրագիր: Դա կարող է մատնել համակարգի գոյության փաստը:

2.2.3 Եզրակացություն դիմակի ընտրման վերաբերյալ

Ժամանակի վրա հիմնված դիմակը պարզ հիշվող է և բավարար անվտանգ ուժային գրոհին այն թույլ է տալիս բոլոր հաշվարկները կատարել մտքում և չի մատնում թաքստոցի գոյության փաստը, սակայն մատնելու դեպքում հակառակորդին, ով գիտի դիմակի ստացման ալգորիթմը, մնում է ֆիքսել նվագված կոմբինացիան կոնկրետ ժամին: Հաշվի առնենք, որ օգտագործողը կարող է շատ տարբեր կոմբինացիաներ նվագել և եթե նրա հեռախոսին հետևող մարդ կա, երբեք չի նվագի դիմակի կոմբինացիան, կամ նա կարող է ուղղակի ականջակալներով նվագել: Այս հետևությունները գործնականում բավարարում են անվտանգության պահանջներին և նշանակում որ նշված դիմակի ստացման ալգորիթմը պիտանի է օգտագործման համար:

Կոորդինատների հիման վրա մշակված ալգորիթմում անհրաժեշտություն էր առաջանում հավելյալ հաշվարկ կատարող ծրագրի և դա մատնում է թաքստոց ունենալու փաստը, կամ ամենաքիչը կասկած է առաջացնում: Սա շատ կարևոր է համակարգի համար այդ իսկ պատճառով այս ալգորթմով ստեղծվող դիմակը չի բավարարում անվտանգության դրված պահանջներին:

Այսպիսով ուսումնասիրելով դիմակի ստեղծման երկու ալգորիթմներ, ի վերջո ընտրվել է ժամանակի վրա հիմնված դիմակի ստացման ալգորիթմը:

2.3 Ֆայլերի հետ աշխատանքի նկարագրություն

Առաջարկվող հավելվածի իրական ինտերֆեյսի հետևում կանգնած է ֆայլերի հետ աշխատանքի գործիքամիջոցները: Ըստ ֆունկցիոնալության դրանք դասակարգվում են երկու խմբի.

- Ֆայլերի ընտրության գործիքամիջոցներ: Վերջիններս անհրաժեշտ են ֆայլերի կարդալու համար
- Նոր ֆայլերի և պանակների ստեղծման ու տեղակայման գործիքամիջոցներ:

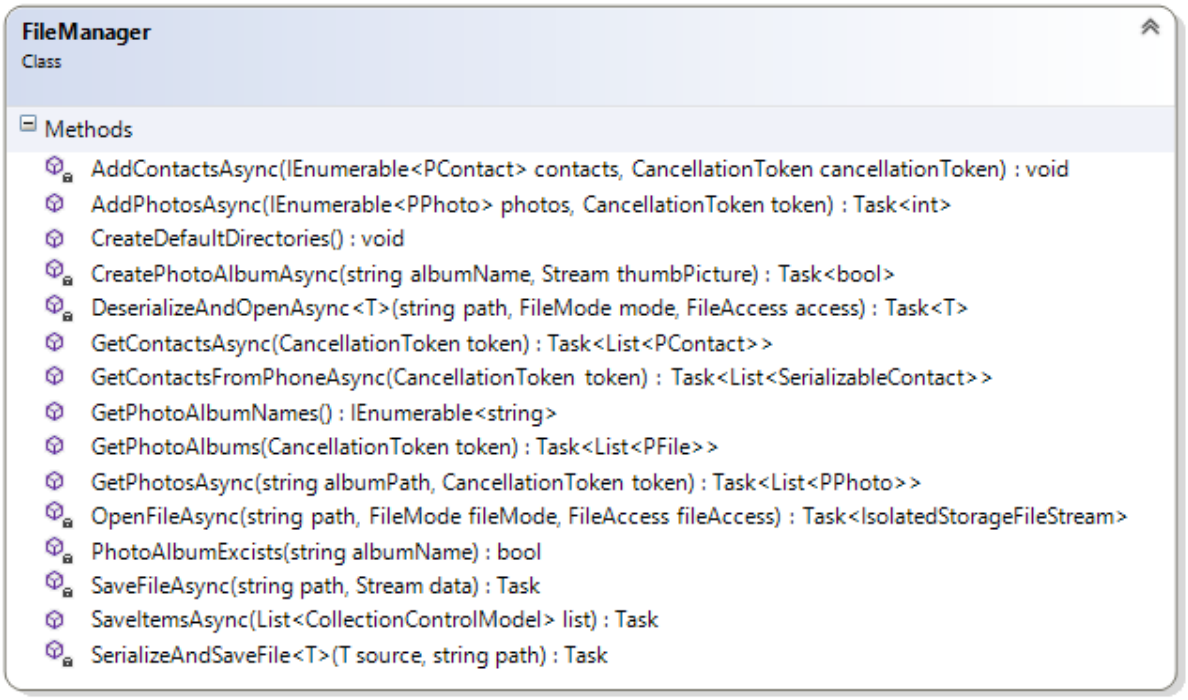
Ըստ գործիքամիջցների աշխատանքի օբյեկտի գործիքամիջոցները դասակարգվում են երկու խմբի.

- Հեռախոսի ներքին ֆայլային համակարգին դիմումներ իրականացնող գործիքամիջոցներ: Դրանք անհրաժեշտ են նկարները և կոնտակտները հեռախոսից ստանալու համար
- Հավելվածի ներքին ֆայլային համակարգին դիմումներ կատարելու գործիքամիջոցներ: Դրանք անհրաժեշտ են հավելվածի ներքին հիշողությունում նկարների և կոնտակտների թաքցման, ինչպես նաև բեռնման ու դիտարկման համար:

Վերը շարադրված ֆունկցիոնալություն ապահովու են FileManager դասի ֆունկցիաները: Վերջիններս կառուցված են ասինխրոն փաթերնի կիրառմամբ:

FileManager դասի կառուցվածքը տրված է նկ. 2.3.1-ում: Դիտարկենք ֆունկցիաները հերթականությամբ:

Task SaveFileAsync(string path, Stream data): Ֆունկցիան ստանում է հիշողության սիմվոլային հասցեն և հոսանքի ֆորմատով ներկայացված ֆայլը, ստեղծում այն, եթե գոյություն չունի և գրանցում տրված հասցեով: Ֆունկցիան ասինխրոն է վերադարձնում է Task, որի վրա կարելի է կիրառել await օպերատորը:



Նկ. 2.3.1 FileManager դասի կառուցվածքը

Task<IsolatedStorageFileStream OpenFileAsync(string path, FileMode fileMode, FileAccess fileAccess): Դիմում է կատարում իզոլացված հիշողությանը: Տրված ճանապարհով ֆայլը գտնելու դեպքում բացում է այն փոխանցված ռեժիմով (Ստեղծել նորը, բացել եթե կա, խափանվել եթե կա) ու թույլտվությամբ(կարդալ, կարդալ ու գրել): Արդյունքը հոսանքի միջոցով վերադարձվում է: Ֆունկցիան ասինխրոն է վերադարձնում է Task, որի վրա կարելի է կիրառել await օպերատորը:

Task SerializeAndSaveFile<T>(T source, string path): Ընդհանրացված տիպի ֆունկցիա է: Կարող է ստանալ ցանկացած սերիալիզացման կոնտրակտներ պարունակող դասի օբյեկտ, սերիալիզացնի այն և կանչի SaveFile տրված հասցեով: Ֆունկցիան ասինխրոն է վերադարձնում է Task, որի վրա կարելի է կիրառել await օպերատորը:

Task<T> DeserializeAndOpenAsync<T>(string path, FileMode fileMode, FileAccess fileAccess): Ընդհանրացված տիպի ֆունկցիա է: Տրված հացեով կատարում է OpenFile գործողությունը, որից հետո դեսերիալիզացնում է T տիպի օբյեկտի մեջ և վերադարձնում այդ օբյեկտը: T տիպը պետք է պարունակի համապատասխան դատա կոնտրակներ: Ֆունկցիան ասինխրոն է վերադարձնում է Task, որի վրա կարելի է կիրառել await օպերատորը:

void AddContactsAsync(IEnumerable<PContact> contacts, CancellationToken token): Սերիալիզացնում և պահում է կոնտակտի մոդելի օբյեկտներ: Ունի չեղարկման ֆունկցիոնալություն, քանի որ այս ֆունկցիայի աշխատանքը կարող է երկար տևել:

Task<int> AddPhotosAsync(IEnumerable<PPhoto> photos, CancellationToken token): Սերիալիզացնում և պահում է նկարների մոդելի օբյեկտներ: Ունի չեղարկման ֆունկցիոնալութուն, քանի որ այս ֆունկցիայի աշխատանքը կարող է երկար տևել: Վերադարձնում է գրանցված ֆայլերի քանակը: Ֆունկցիան ասինխրոն է վերադարձնում է Task, որի վրա կարելի է կիրառել await օպերատորը:

Task SaveItemsAsync(List<CollectionControlModel> list): Այս ֆունկցիան արտաքին ինտերֆեյսի համար և FileManager-ին դիմելու ինտերֆեյսն է: Այն ստանում է մոդելի օբյեկտներ, տեսակավորում դրանց ըստ տիպի՝ նկար, ալբոմ, կոնտակտ , կախված այդ տիպից կազմավորում է պահպանման հասցեն և համապատասխանաբար կանչում AddPhotosAsync կամ AddContactsAsync ֆունկցիան ֆունկցիան՝ այդ հասցեով պահելով բոլոր ֆայլերը: Ֆունկցիան ասինխրոն է վերադարձնում է Task, որի վրա կարելի է կիրառել await օպերատորը:

Task<bool> CreatePhotoAlbumAsync(string albumName, Stream thumbPicture): Ստեղծում է ֆոտո ալբոմի համար պանակ և պահպանում ալբոմի մասին ինֆորմացիան ներառյալ ալբոմի նկարագրիչ նկարը: Եթե այդիպսի ալբոմ արդեն գոյություն ունի, գործողությունը չի կատարվում և վերադարձվում է false, հակառակ դեպքում true: Ֆունկցիան ասինխրոն է վերադարձնում է Task, որի վրա կարելի է կիրառել await օպերատորը: void CreateDefaultDirectories(): Իզոլացված հիշողությունում ստեղծում է ալբոմների, կոնտակտների, նրանց նկարագրիչ նկարների համար անհրաժեշտ պանակները: Եթե դրանք արդեն գոյություն ունեն, վերադառնում է առանց որևէ բան անելու:

bool PhotoAlbumExcists(string albumName): Վերադարձնում է true, եթե նման անունով ալբոմ արդեն կա իզոլացված հիշողությունում, հակառակ դեպքում վերադարձնում է false:

Task<List<PContact> GetContactsAsync(CancellationToken token): Իզոլացված հիշողությունից կարդում է բոլոր պահված կոնտակները, դեսերիալիզացնում և վերադարձնում PContact զանգվածի տեսքով: Ունի չեղարկման հնարավորություն: Ֆունկցիան ասինխրոն է վերադարձնում է Task, որի վրա կարելի է կիրառել await օպերատորը:

Task<List<SerializableContact>> GetContactsFromPhoneAsync(CancellationToken token): Հեռախոսի հիշողությունից բեռնում է բոլոր կոնտակտները և դրանց պարունակությունը սերիալիզացնում SerializableContact դասի օբյեկտների մեջ: Վերադարձնում է վերջիններիս զանգվածը: Ունի չեղարկման հնարավորություն: Ֆունկցիան ասինխրոն է վերադարձնում է Task, որի վրա կարելի է կիրառել await օպերատորը:

IEnumerable<string> GetPhotoAlbumNames(): Վերադարձնում է իզոլացված հիշողությունում գտնվող նկարների ալբոմների անունները:

Task<List<PFile>> GetPhotoAlbums(CancellationToken token): Դիմում է իզոլացված հիշողությանը, ստանում բոլոր ալբոմների ֆայլերը, դեսերիալիզացնում PFile դասի և վերադարձնում դրանց: Ունի չեղարկման հնարավորություն: Ֆունկցիան ասինխրոն է վերադարձնում է Task, որի վրա կարելի է կիրառել await օպերատորը:

Task<List<PPhoto>> GetPhotosAsync(string albumPath, CancellationToken token): Դիմում է իզոլացված հիշողությանը, ստանում տվյալ ալբոմի բոլոր նկարների ֆայլերը, դեսերիալիզացնում և վերադարձնում դրանց: Ունի չեղարկման հնարավորություն: Ֆունկցիան ասինխրոն է, վերադարձնում է Task, որի վրա կարելի է կիրառել await օպերատորը:

Գլուխ 3: Ծրագրի աշխատանքը

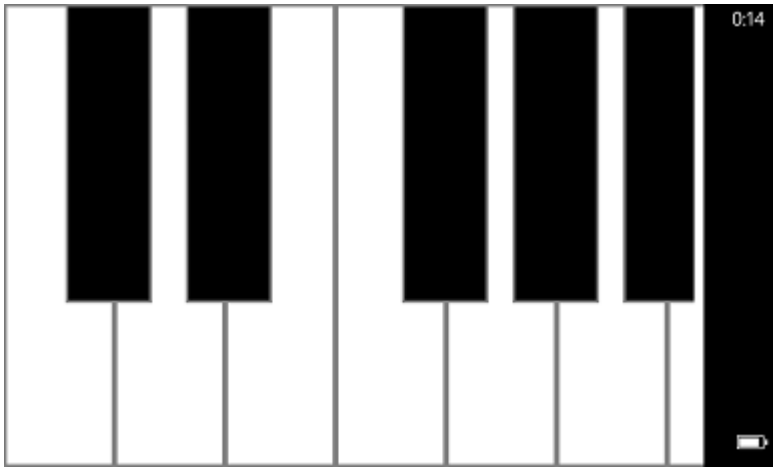
My Piano անձնական տեղեկատվության թաքցման ծրագիրը, ստեղծված է Windows Phone 8 օպերացիոն համակարգի համար: Այն բաղկացած է, երկու ինտերֆեյսրից՝ կեղծ

և իրական: Կեղծ ինտերֆեյսը ներկայանում է դաշնամուրի հավելվածի տեսքով: Իրական ինտերֆեյսը բաղկացած է հետևյալ բաժիններից.

- Թաքցված ալբոմների դիտարկիչ
- Թաքնված կոնտակների դիտարկիչ
- Նկարի մեծացման, շարժման ֆունկցիոնալությամբ դիտարկիչ
- Կոնտակտի դետալների դիտարկիչ
- Հեռախոսի ալբոմների և նկարների ընտրման էջ
- Հեռախոսի կոնտակների ընտրման էջ
- Գաղտնաբառի փոփոխման էջ

Այժմ անդրադառնանք բոլոր բաժիններին հերթականությամբ:

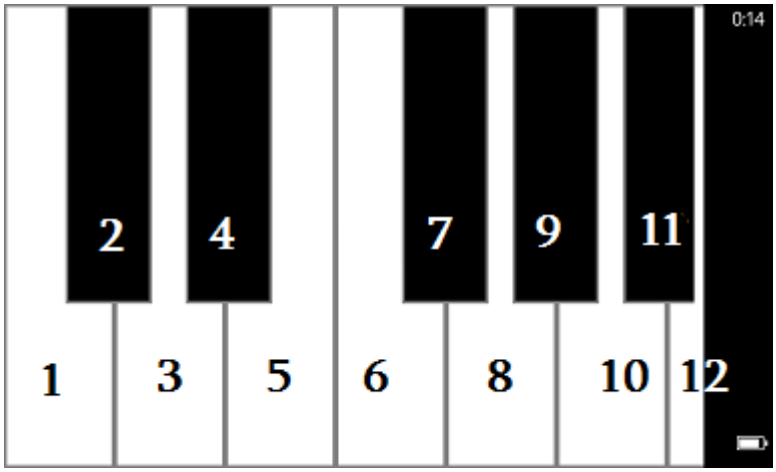
Կեղծ ինտերֆեյս: Երբ բացում ենք ծրագիրը, բացվում է հորիզոնական ուղղվածությամբ դաշնամուր պարունակող էջ (նկ. 3.1)



Նկ. 3.1 Կեղծ ինտերֆեյս: Մեկ օկտավա պարունակող դաշնամուր

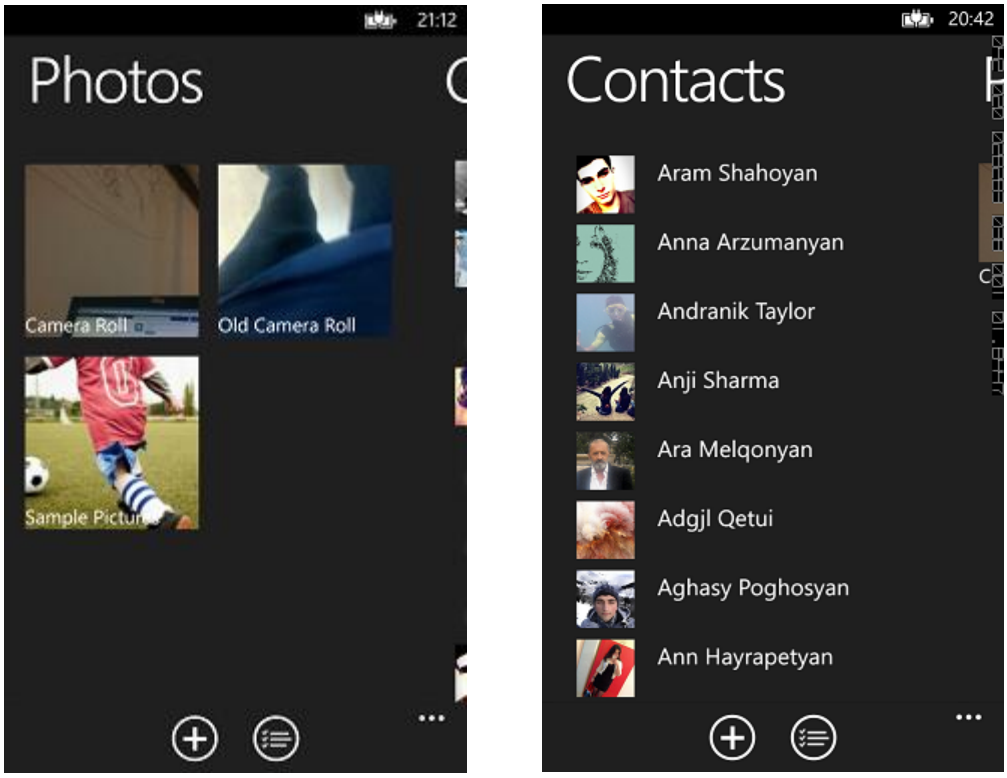
Դաշնամուրի ստեղները սեղմելիս նվագարկվում են համապատասխան հնչյունային սեմփլները, որոնք մշակվել են FL Studio 10 ծրագրային սպասարկման միջոցով: Մշակման ընթացքում բելեր սեմփլների ալիքները հարթեցվել են և մշակվել բացառապես տվյալ աշխատանքի շրջաններում և այդպիսով չի խախտվել այլ անձանց հեղինակային իրավունքներ: Կարելի է երկար նվագել, առանց նույնիսկ կասկածելու իրական ինտերֆեյսի գոյության մասին: Դրան է նաև նպաստում այն փաստը, որ կեղծ ինտերֆեյսի հետ աշխատելիս չի կատարվում իրական ինտերֆեյսի բեռնման և ոչ մի

գործողություն: Էկրանի աջ կողմում երևում է ժամացույցը և մարտկոցի լիցքավորման քանակը: Սա սովորական մոտեցում է Windows Phone հավելվածների համար: Ըստ դիմակի ալգորիթմի, ժամացույցի ցուցման ամեն արժեքին գումարելով գաղտնաբառի համապատասխան արժեքը ստանում ենք հերթականություն, որը ձախից սկսված 1-13 համարակալելով կարող ենք նվագել դաշնամուրի վրա (նկ. 3.2): Ճիշտ դիմակը, ինչպես գիտենք փոփոխվում է ամեն 1 րոպեն մեկ, այդ իսկ պատճառով ավելի հարմար կլինի նախապես հաշվարկել հաջորդ րոպեի դիմակը, որպեսզի հաշվարկից հետո ժամանակ ունենանք այն նվագելու:



Նկ. 3.2 Դաշնամուրի վրա ստեղները մտովի համարակալվում են 1-12

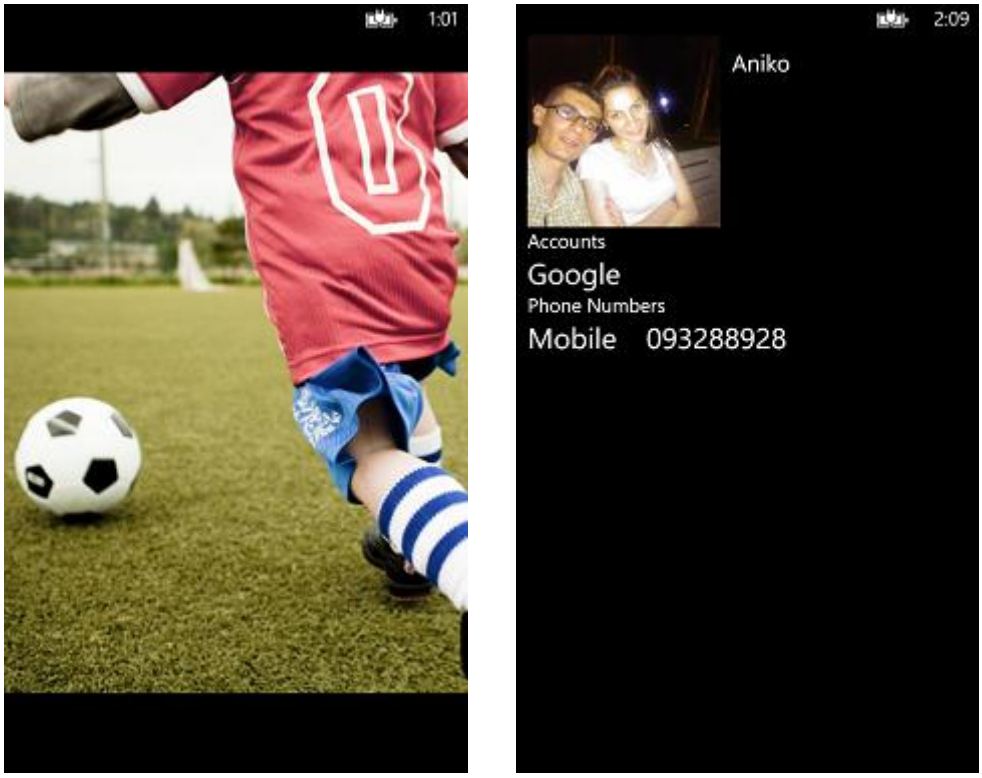
Իրական ինտերֆեյս: Երբ նվագում ենք դիմակը, բացվում է իրական ինտերֆեյսի առաջին էջը, որտեղ տեղադրված է Pivot տեսակի օբյեկտ: Pivot-ը թույլ է տալիս ունենալ հորիզոնական էջավորված, վերնագիր ունեցող, գլորվող անիմացիայով կոնտենտներ: Էջը բացվելիս ծրագրի ինտերֆեյսը դիմում է մոդելին տվյալներ ստանալու պահանջով: Մոդելը ճանաչելով, որ կանչողը թաքուն ֆայլերի ցանկն է, դիմում է FileManager դասին, որը կատարում է դիմում ֆայլային համակարգին՝ այնտեղ փնտրելով, նկարների ալբոմներ և կոնտակներ: Գտնված արժեքները ետ են վերադարձվում մոդելին, որը դրանք փոխանցում է ինտերֆեյսին և օգտագործողը տեսնում է իր ընթացիկ թաքուն տվյալները (նկ. 3.3):



Նկ. 3.3 Թաքուն ֆայլերի դիտարկման էջ

Որևէ ալբոմ ընտրելիս, նույն տեղում հայտնվում են ալբոմի նկարները: Այդ նկարներից ցանկացածի վրա սեղմելիս բացվում է նկարի դիտարկման էջը (նկ. 3.4): Այստեղ կարող ենք օգտագործել մատով շարժելու, երկու մատներով մեծացնելու և փոքրացնելու ֆունկցիոնալությունները: Ետ վերադառնալով կարող ենք բացել կոնտակների էջից որևէ կոնտակտ և կբացվի, կոնտակների դիտարկման էջը: Այստեղ տեսակավորված տեսքով երևում է օգտագործողի նկարը, անունը, ազգանունը, միջին անունը, կոչումը, սոցիալական կայքերի տվյալները, ընտանիքի անդամների տվյալները, աշխատավայրերի տվյալները, բնակման հասցեն, մարզը, երկիրը, ռեգիոնը, քաղաքը, հեռախոսահամարները և դրանց տեսակները և նշումներ:

Նկարագրված դիտարկիչների շնորհիվ տվյալ աշխատանքը իրենից ներկայացնում է ոչ միայն տվյալների թաքցման միջոց, այլև թույլ է տալիս դիտարկել այդ տվյալները թաքցման ժամանակ:



Նկ. 3.4 Նկարների և կոնտակների դիտարկիչներ

Ինչպես երևում է էջի ներքևում կան փոքրիկ կլոր կոճակներ: Դրանք կազմում են Application Bar կոչվող մենյուի մի մասը և ընդունված են ընտրման և այլ գործողություններ գորշարկելու համար: Երբ սեղմում ենք առաջին կոճակը բացվում է ընտրման ցանկի էջը (Նկ. 3.5): Այն նույնպես բաղկացած է Pivot-ից: Էջը բացվելիս ինտերֆեյսը դիմուն է կատարում մոդելին, որը դիմում է FileManager-ին, որը ճանաչելով դիմողին հարցում է կատարում հեռախոսի ներքին ֆայլային համակարգին գոյություն ունեղող նկարների ալբոմների և կոնտակների համար: Ստացված արդյունքը FileManager-ը վերադարձնում է մոդելին, որն էլ այն փոխանցում է ինտերֆեյսին՝ ցուցադրելով բոլոր ֆայլերը ցանկի տեսքով: Ցանկից կարելի կատարել բազմակի ընտրություն, կամ եթե գործ ունենք ֆոտո ալբոմի հետ, կարելի է բացել այդ ալբոմը և ընտրել ներքին նկարենրից: Նկարներ և կոնտակտներ ընտրելուց հետո պետք է սեղմել ներքևի Application Bar-ի Complete կոճակին: Այդ կոճակը սեղմելիս բոլոր ընտրված ֆայլերը փոխանցվում են ֆայլերի մենեջերին պահպանման համար: Մենեջերը նախ ստուգում է արդյոք այդ ֆայլերը արդեն իսկ գոյություն չունեն, կրկնությունները բացառելու համար և չկչկնվելու դեպքում սերալիզացնում է նկարները և կոնտակները

և պահպանում իզոլացված հիշողության համապատասխան բաժնում, որտեղից հետագայում կարդալու է այդ ֆայլերը: Ընտրության ընթացքում կարելի է մտնել ընտրման ռեժիմ, երբ ներքև կոճակը select ֆունկիոնալ կոճակն է և ավարտել ընտրությունը՝ այն չեղարկելով, որի համար պետք է սեղմել հետ գնալու կոճակը կամ էլ պահպանել սեղմելով completed կոճակը:

Նկ. 3.5 Նկարների և կոնտակտների ընտրման ցակները

Գլուխ 4: Ծրագրի տնտնեսական հիմնավորում

Ներդրումային օբյեկտը ծրագրային համակարգ է: Ձեռքբերման հետ կապված տվյալները ներկայացված են հետևյալ աղյուսակում, որի հիման վրա կատարվում է հաշվարկը:

4.1 Ծրագրի ձեռք բերման կապիտալ ներդրումների հաշվարկը

Ձեռնարկությունների հիմնական միջոցների ստեղծման, ընդլայնման և տեխնիկական վերազինման նպատակով կատարված ծախսերը կոչվում են կապիտալ

ներդրումներ: Կապիտակ ներդրումները ձեռնարկության գործունեության զարգացման հիմքն են կազմում, որոնք առավելապես իրականացվում են իրական ներդրումների ձևով:

Կապիտալ ներդրումների իրականացման ձևերն են՝

1. Նոր ձեռնարկությունների շինարարությունը,
2. Գործող ձեռնարկությունների տեխնիկական վերազինումը,
3. Գործող ձեռնարկությունների վերակառուցումը,
4. Գույքային համալիրի ձեռքբերումը:

Նոր ձեռնարկությունների շինարարությունը պահանջում է ժամանակակից արտադրություն ստեղծելու համար կապիտալ ներդրումների իրականացում, որոնք ապահովում են բարձր արտադրանքի արտադրողականություն և բավարարում են էկոլոգիական անվտանգության պահանջներին:

Գույքային համալիրի կապիտալ ներդրումները իրականացվում են արտադրական և տնտեսական նպատակներով օգտագործվող գույքի և տրանսպորտային միջոցների ձեռքբերման համար:

Կապիտալ ներդրումները բաժանվում են՝

1. Արտադրական կապիտալ ներդրումների, որտեղ ներդրումային օբյեկտներ են հանդիսանում արտադրական նշանակության հիմնական միջոցները՝ արտադրական շենքերը, կառուցվածքները, սարքավորումները և այլն:
2. Ոչ նյութական ակտիվներում ներդրումների ՝ այսինքն կապիտալ ներդրումների ինտելեկտուալ սեփականության մեջ, որտեղ ներդրումային օբյեկտեներ են հանդիսանում գույքային իրավունքը, արտոնագիրը, գիտական աշխատանքը, արտադրության տեխնոլոգիան, ծրագրային ապահովումը և այլն:

Նոր տեխնիկային ձեռքբերման կամ արտադրության համար ձեռնարկությունը պետք է կատարի կապիտալ ներդրումներ: Կապիտալ ներդրումների միջոցով հնարավորություն է ստեղծվում ներկա ժամանակում զարգացման ծրագրերի իրականացման համար որոշակի գումարներ ծախսել՝ հետագայում ավելի մեծ գումարներ ստանալու նպատակով:

Կապիտալ ներդրումների արժեքը և տնտեսական արդյունավետությունը հաշվարկվում է մինչև այդ ներդրումների իրագործումը, այսինքն ներդրումային ծրագրերի իրականացման փուլում: Եթե նախագծման փուլում պարզվում է, որ ներդրումային նախագիծը տնտեսապես ձեռնտու է, ապա այն իրագործվում է, հակառակ դեպքում չի իրագործվում:

Նոր սարքավորուման կապիտալ ներդրումները կարող են իրականացվել սարքավորման գնման միջոցով: Սարքավորման գնման դեպքում կապիտալ ներդրումները որոշվում են հետևյալ բանաձևով,

$$K_q = P(1 + \sigma_1 + \sigma_2),$$

Որտեղ $P - Ն$ սարքավորման գնման գինն է,

$\sigma_1 - Ը$ Վաճառքի հաշվարկը որոշող գործակիցը $(\sigma_1 = 0.2)$,

$\sigma_2 - Ը$ Սարքավորման տեղափոխման տրանսպորատային ծախսերի որոշման գործակիցը $(\sigma_2 = 0.05 - 0.1)$:

Եթե սարքավորումը ձեռք է բերվում արտասահմանյան երկրից, ապա կապիտալ ներդրումները որոշվում են՝

$$K_q = P \cdot (1 + \sigma_1 + \sigma_2 + \sigma_3),$$

Որտեղ ՝ $\sigma_3 - Ը$ ներմուծման մաքսատուրքի գործակիցն է $(\sigma_3 = 0{,}1 - 0{,}15)$:

Եթե սարքավորումը շահագործման մեջ մտցնելու համար պահանջվում են շինարակական և տեղակայաման աշխատանքներ, ապա կապիտալ ներդրումների որոշման բանաձևը կընդունի հետևյալ տեսքը,

$$K_q = P \cdot (1 + \sigma_1 + \sigma_2 + \sigma_3 + \sigma_4 + \sigma_5)$$

Որտեղ $\sigma_4 - Ը$ գործակից է, որը հաշվի է առնում շինարարական աշխատանքների արժեքը $(\sigma_4 = 0{,}02 - 0{,}03)$

$\sigma_5 - Ը$ սարքավորման տեղակայման ծախսերի որոման գործակիցն է $(\sigma_5 = 0{,}05 - 0{,}1)$:

Եթե սարքավորումը շահագործման հանձնելու համար շինարարական և տեղակայման աշխատանքներ չեն պահանջվում, ապա բանաձևի մեջ σ_4 և σ_5 գործակիցները պետք է վերցնել հավասար զրոյի:

$$K_q = P \cdot (1 + \sigma_1 + \sigma_2 + \sigma_3 + \sigma_4) = 200000 \cdot (0.2 + 0.1 + 0.15 + 0.02) = 94000 \text{ դրամ}$$

4.2 Ծրագրի նախագծման և արտադրության կապիտալ ներդրումների հաշվարկ

Սարքավորումների նախագծային ինքնարժեքը որոշվում է տեսակարար կշիռների մեթոդով, որի էությունն այն է, որ, իմանալով ինքնարժեքի կազմում ուղղակի արտադրական ծախսերից որևէ մեկի մեծությունը և դրա տեսակարար կշիռը, ինքնարժեքի մեջ կարելի է որոշել արտադրական ինքնարժեքը:

$$C_ա = L \cdot \frac{100}{d},$$

$C_ա$ -ն սարքավորման նախագծային ինքնարժեքն է,

L-ը սարքավորման արտադրության համար վճարված աշխատավարձի գումարն է ,

d-ն սարքավորման ինքնարժեքի մեջ աշխատավարձի տեսակարար կշիռն է ՝ արտահայտված տոկոսներով:

Աշխատավարձի ֆոնդը բաղկացած է հիմնական և լրացուցիչ աշխատավարձից,

$$L = (L_հ + L_լ) \cdot m.$$

$L - ը$ աշխատավարձի ֆոնդն է ,

$L_հ - ը$ հիմնական աշխատավարձի ամսական ֆոնդը,

$L_լ - ը$ լրացուցիչ աշխատավարձի ամսական ֆոնդը

m-ը սարքավորման նախագծման և արտադրության համար պահաջվող ժամանակը ՝ ամիսներով:

$$C_ա = L \cdot \frac{100}{d} = (L_հ + L_լ) \cdot m \cdot \frac{100}{d} = \left(80000 + 80000 \cdot \frac{10}{100}\right) \cdot 1 \cdot \frac{100}{95} = 92632 \text{ դրամ}$$

4.3 Թողարկվող արտադրանքի ինքնարժեքի հաշվարկը

Արտադրանքի ինքնարժեքը նպատակահարմար է նախ որոշել տարվա կտրվածքով, ապա այն բաժանելով արտադրանքի թողարկման տարեկան ծավալին որոշել միավորի ինքնարժեքը: Արտադրանքի ինքնարժեքը որոշվում է.

$$C= M + L + L_u + H, \text{ որտեղ}$$

M-ը արտադրանքի արտադրության համար պահանջվող նյութական ծախսերն են,

L-ը արտադրանքի արտադրության համար վճարված աշխատավարձի ծախսերը,

L_u -ը սոցիալական ապահովագրական վճարումների ծախսերը,

H-ը անուղղակի արտադրական ծախսերը:

Նյութական ծախսերը ներառում են արտադրական նպատակով ձեռք բերվող նյութերի արժեքը, որոնք մտնում են թողարկվող արտադրանքի ինքնարժեքի մեջ: Նյութական ծախսերի կազմի մեջ արտացոլվում են նաև գնովի կիսապատրաստուկների, համալող արտադրանքի և արտադրական նպատակով օգտագործվող էլեկտրաէներգիայի արժեքները:

- Նյութական ծախսեր

$$M = C_q + E$$

C_q-ը արտադրանքի արտադրության կամ ծառայությունների մատուցման համար պահանջվող գնովի նյութերի արժեքն է,

E -ը արտադրանքի արտադրության կամ ծառայությունների մատուցման համար պահանջվող էլեկտրաէներգիայի ծախսը:

$C_q = 2000$ *դրամ*

$K_g = K_z = 0$

$$W = \frac{N \cdot F \cdot K_u \cdot K_p}{(1 - K_g) \cdot (1 - K_z)} = \frac{0.05 \cdot 245 \cdot 8 \cdot 0.85 \cdot 0.9}{(1-0)(1-0)} = 74.97$$

$$E = W \cdot 30 = 74.97 \cdot 30 = 2249.1 \text{ } \textit{դրամ}$$

$$M = 2000 + 2249.1 = 4250 \text{ } \textit{դրամ}$$

- Աշխատանքի վճարման ծախսեր

Աշխատանքի վճարման ծախսերի հաշվարկը կատարվում է՝ ելնելով ներդվող սարքավորման շահագործումն իրականացնող բանվորների աշխատավարձերի գումարներից:

$$L = \left(L_հ + L_լ\right) \text{ որտեղ}$$

$L_հ$-ը հիմնական աշխատավարձի ամսական ֆոնդն է,

$L_լ$ -ը լրացուցիչ աշխատավարձի ամսական ֆոնդը:

$$L = \left(L_հ + L_լ\right) \cdot 12 = \left(80000 + 80000 \cdot \frac{10}{100}\right) \cdot 12 = 1056000 \textit{ դրամ}$$

- Սոցիալական ապահովագրական վճարներ

Պարտադիր սոցիալական ապահովագրական վճարները տվյալ ձեռնարկության համար դիտարվում է որպես ծախս, որը ցույց է տրվում թողարկվող արտադրանքի ինքնարժեքի մեջ:

$$L_1 = 80000 \cdot 24.4\,\% = 19520 \textit{ դրամ}$$

$L_2 = 0$

վճարների տարեկան գումարը

$$L_ս = 19520 \cdot 12 = 234240 \textit{ դրամ}$$

Անուղղակի արտադրական ծախսեր

Անուղղակի արտադրական ծախսերը այնպիսի ծախսեր են, որոնք պարբերաբար հատուկ մեթոդներով բաշխվում են պատրաստվող արտադրանքի ծախսերի վրա: Այդ ծախսերից են՝ արտադրական շենքերի ամորտիզացիան, սարքավորումների ամորտիզացիան, սարքավորումների ընթացիկ վերանորոգումների ծախսերը և այլ անուղղակի ծախսեր:

$$H_ս = H_1 + H_2 + H_3 + H_4 \text{ որտեղ}$$

H-ը անուղղակի արտադրական ծախսերի գումարն է,

H_1 –ը արտադրական շենքի ամորտիզացիան,

H_2 -ը սարքավորումների ամորտիզացիան,

H_3 -ը սարքավորումների ընթացիկ վերանորոգումների ծախսեր:

H_4 -ը անուղղակի արտադրական այլ ծախսեր:

$$H_1 = \frac{b \cdot \beta}{T} = 7 * \frac{140000}{10} = 98000 \text{ դրամ}$$

$$H_2 = \frac{S_u}{T} = \frac{92632}{10} = 9263 \text{ դրամ}$$

$$H_3 = \frac{S_u \cdot q \cdot \alpha_2}{100} = \frac{9263 \cdot 3 \cdot 0.01}{10} = 278 \text{ դրամ}$$

$$H_4 = L \cdot 0.1 = 1056000 \cdot 0.1 = 105600 \text{ դրամ}$$

$$H = 98000 + 9263 + 278 + 105600 = 213140 \text{ դրամ}$$

- Արտադրական ինքնարժեք

$$C = M + L + L_u + H = 4250 + 1056000 + 234240 + 213140 = 1507630 \text{ դրամ}$$

- Արտադրանքի միավորի արտադրական ինքնարժեք

$$C_u = \frac{C}{1500} = \frac{1507630}{1500} = 1005 \text{ դրամ}$$

- Արտադրանքի լրիվ ինքնարժեք

Արտադրանքի իրացումից ֆինանսական արդյունքը որոշելու համար արտադրական իքնարժեքի հետ միասին պետք է հաշվի առնել նաև իրացման և վարչական ծախսերը: Արտադրական ինքնարժեքը իրացման և վարչական ծախսերի հետ միասին կազմում են արտադրանքի լրիվ ինքնարժեքը:

$C_l = C + C_h + C_վ$ որտեղ

C_l-ը արտադրանքի լրիվ ինքնարժեքն է,

C –ը արտադրանքի արտադրական ինքնարժեքը,

C_h-ը արտադրանքի իրացման ծախսերն են,

$C_վ$ -ը վարչական ծախսերը:

$$C_{լ} = C + C_{ի} + C_{կ} = C + C \cdot \frac{\alpha_3}{100} + C \cdot \frac{\alpha_4}{100} = 1507630 \cdot \left(1 + \frac{25}{100} + \frac{15}{100}\right) = 2110682 \ \textit{դրամ}$$

- Արտադրանքի միավորի լրիվ ինքնարժեք

$$C_{միլ} = \frac{C_{լ}}{1500} = \frac{2110682}{1500} = 1407 \ \textit{դրամ}$$

4.4 Ներդրումներից եկամուտներ և NPV-ի հաշվարկը

Ներդրումային նախագծերի գնահատման համար գոյություն ունեն տարբեր մեթոդներ , որոնք բաժանվում են երկու խմբի: Առաջին խմբի մեջ մտնում են այն մեթոդները, որոնք հաշվի են առնում ժամանակի գործոնը և կոչվում են դիսկոնտային մեթոդներ: Այդ մեթոդի դեպքում հաշվարկների կատարման ժամանակ օգտագործվում են եկամուտների և ծախսերի արժեքները:

Դիսկոնտային մեթոդների թվին են դասվում՝

1. Մաքուր բերված արժեքը ՝ NPY,
2. Շահույթի ներքին նորմը՝ IRR,
3. Հետգնման դիսկոնտային ժամկետը ՝ DPP:

Երկրորդ խմբի մեջ մտնում են այն մեթոդները, որոնք հաշվի չեն առնում ժամանակի գործոնը և կոչվում են հաշվապահական մեթոդներ: Հաշվապահական մեթոդներն են՝

1. Ներդրումների արդյունավետության գործակիցը՝ ARR,
2. Ներդրումների հետգնման ժամկետը՝ PP:

Պարզության համար ընդունենք, որ ներդրումից եկամուտների ստացումը կատարվում է հավասարաչափ: Այդ դեպքում եկամուտների որոշման համար կունենանք

$$P = Q \cdot (X - C) \cdot (1 - t) = 1500 \cdot (2000 - 1407) \cdot (1 - \frac{20}{100}) = 711600 \ \textit{դրամ}$$

NPV-ի հաշվարկի համար օգտվենք (5.3) բանաձևից

Եթե ներդրումները իրականացվում են n_1 տարիների ընթացքում, իսկ $(n_1 + 1)$-րդ տարվանից ստացվում են եկամուտները, ապա մաքուր բերված արժեքը որոշվում է.

$$NPV = \sum_{j=1}^{n_2} \frac{P_j}{(1+r)^{n_1+j}} + \sum_{i=1}^{n_1} \frac{K_i}{(1+r)^i}$$

P_j-ը ներդրումներից ստացվող եկամուտն է i-րդ տարում,

n_1-ը ներդրումների իրականացման տարիների թիվը,

n_2-ը եկամուտների ստացման տարիների թիվը,

K_i-ը ներդրումային ծախսերը i-րդ տարում

r-ը դիսկոնտի դրույքաչափը:

Եթե NPV>0 ,ներդրումային նախագիծը ընդունվում է,

NPY<0 , նախագիծը չի ընդունվում,

NPY=0, նախագիծը ոչ եկամտաբեր է և ոչ էլ վնասաբեր:

$$NPV = P \cdot \frac{1 - (1+r)^{-n_2}}{r \cdot (1+r)} - \frac{K}{1+r} = 711600 \cdot \frac{1 - \left(1 + \frac{15}{100}\right)^{-3}}{\frac{15}{100} \cdot \left(1 + \frac{15}{100}\right)} - \frac{94000}{1 + \frac{15}{100}} = 1327229 \text{ դրամ}$$

Ներդրումների հետգնման ժամկետը հավասար է տարիների այն թվին, որի դեպքում ներդրումներից ստացված եկամուտը հավասար է դառնում ներդրումային ծախսերին, այսինքն, երբ NPV = 0:

Բանաձևից որոշված n_2 –ը կլինի ներդրումների հետգնման ժամկետը: Քանի որ

$$\frac{200}{1.15^3} + \frac{300}{1.15^4} < \frac{300}{1.15} + \frac{200}{1.15^2}$$

հետևաբար ներդրումների ժամկետը 4 տարի է:

Այժմ ընդունենք, որ ներդրումից եկամուտների ստացումը կատարվում է անհավասարաչափ, որը պետք է հաշվարկվի հետևյալ տվյալների համաձայն

Տարի	Q_j (հատ)	X_j (դրամ)	C_j (դրամ)
1	1500	2000	1368
2	1600	2150	1295
3	1550	2300	1236

$$P_1 = Q_1 \cdot (X_1 - C_1) \cdot (1 - t) = 1500 \cdot (2000 - 1368) \cdot (1 - \frac{20}{100}) = 805800 \text{ դրամ}$$

$$P_2 = Q_2 \cdot (X_2 - C_2) \cdot (1 - t) = 1600 \cdot (2150 - 1295) \cdot (1 - \frac{20}{100}) = 1162800 \text{ դրամ}$$

$$P_3 = Q_3 \cdot (X_3 - C_3) \cdot (1 - t) = 1500 \cdot (2000 - 1368) \cdot (1 - \frac{20}{100}) = 1401820 \ \textit{դրամ}$$

Այս դեպքում NPV-ի որոշման համար պետք է օգտվենք (5.2) բանաձևից

$$NPV = \sum_{j=1}^{4} \frac{P_j}{(1+r)^{j+1}} + \frac{K}{1+r} = \frac{805800}{(1+0.15)^2} + \frac{1162800}{(1+0.15)^3} + \frac{1401820}{(1+0.15)^4} - \frac{94000}{1+0.15}$$
$$= 2093624 \ \textit{դրամ}$$

4.5 Հաշվարկի տվյալները

Տվյալների անվանումը	Նշանակումը	Միավորը	Մեծությունը
Սարքավորման ձեռք բերման ներդրումների հաշվարկի տվյալներ			
Սարքավորման գնման գինը	P	դրամ	200000
Վաճառքի հարկը որոշող գործակից	σ_1		0.2
Տրանսպորտային ծախսերի գործակից	σ_2		0.1
Ներմուծման մաքսատուրքի գործակից	σ_3		0.15
Տեղակայման ծախսերի գործակից	σ_4		0.02
Սարքավորման նախագծման և արտադրության ներդրումների հաշվարկի տվյալներ			
Հիմնական բանվորի աշխատավարձ	A_1	դրամ	80000
Օժանդակ բանվորի աշխատավարձ	A_2	դրամ	0
Լրացուցիչ աշխատավարձի որոշման տոկոս	α_1		10
Սարքավորման արտադրության ժամանակը	m	ամիս	1
Աշխատավարձի տեսակարար կշռի տոկոս	d	%	95
Թողարկվող արտադրանքի ինքնարժեքի հաշվարկի տվյալներ			
Գնովի նյութերի ծախսը	Cq	դրամ	2000
Հիմնական բանվորի աշխատավարձ	A_3	դրամ	80000
Օժանդակ բանվորի աշխատավարձ	A_4	դրամ	0
Արտադրական տարածքի մակերեսը	b	մ2	7
Սարքավորման ամորտիզացիայի ժամկետը	T	տարի	10
Սարքավորման հզորությունը	N	ԿՎտ	0.05

1 ԿՎժ էլեկտրաէներգիայի սակագինը	γ	դրամ	38
Ծրագրում հայտնաբերված սխալների թիվը	q	հատ	3
1 նորոգման արժեքը	α_2	%	1
Իրացման ծախսերի որոշման տոկոս	α_3	%	25
Վարչական ծախսերի որոշման տոկոս	α_4	%	15
Սարքավորման ներդրումից եկամուտների և NPV-ի հաշվարկի տվյալներ			
Թողարկվող արտադրանքի տարեկան ծավալը	Q	հատ	1500
Արտադրանքի միավորի գինը	X	դրամ	2000
Ներդրումների իրականացման տարիների թիվը	n_1	տարի	1
Եկամուտների ստացման տարիների թիվը	n_2	տարի	3
Դիսկոնտի տոկոսադրույքը	r	%	15
Շահութահարկի տոկոսադրույքը	t	%	20

4.6 Եզրակացություն

Հաշվարկների ընթացքում հանգեցինք այն եզրակացությանը, NPV-ն որ կազմում է 2093624 դրամ: Քանի որ $NPV > 0$ ապա նախագիծը ընդունվում է:

Գլուխ 5 Կենսագործունեության անվտանգություն

5.1 Կենսագործունեության միջավայրում նախագծվող օբյեկտներից առաջացող հնարավոր վտանգավոր և վնասակար գործոնների վերլուծությունը և դրանց վերացման կամ նվազեցման միջոցառումները

ԷՀՄ-ներով համալրված սենյակներում վտանգավոր գործոններ կարող են հանդիսանալ հետևյալ գործոնները[a]

1. Աղմուկի բարձր մակարդակը
2. Էլեկտրական հոսանքը
3. Հրդեհները
4. Տարածքի արհեստական լուսավորվածության անբավարարվածությաւոնը

5. Բնական լուսավորվածության անբավարարվածությունը կամ բացակայությունը:

6. Ճառագայթումը

1. Աղմուկը աշխատանքային սենյակում լայն տարածված վտանգավոր գործոններից է, որի ազդեցությունը չի սահմանափակվում միայն լսողական օրգանների վրա, նյարդային համակարգի միջոցով, այն ազդում է նաև ներքին օրգանների վրա: Բարձր աղմուկի պայմաններում աշխատող մարդիկ բողոքում են գլխացավերից, գերհոգնածությունից, անքնությունից, որն էլ դառնում է աշխատանքի արտադրողականության անկման պատճառ:

Աղմուկի դեմ պայքարի արդյունավետ միջոցներից է ձայնակլանիչ հատկություններով օժտված նյութերի օգտագործումը: Փակ տարածության մեջ ձայնի ալիքները հասնում են պատերին, առաստաղին կամ այլ խոչընդոտներին, որոնք ոչ միայն կլանում այլև անդրադարձնում են ձայնային էներգիան: Աղմուկից պաշտպանվելու համար գոյություն ունեն անհատական միջոցներ հակաաղմուկային ականջակալներ, խցաններ և այլն:

Աղմուկի աղբյուրներ կարող են հանդիսանալ տվյալ և հարակից տարածքներում աշխատող էլեկտրական, էլեկտրոնային և մեխանիկական սարքեր և սարքավորումներ: Օրինակ օդափոխիչները, տպիչները և այլն:

2. Էլեկտրական սարքավորումները, որոնց թվին են պատկանում ԷՀՄ-ի բոլոր սարքերը մարդու համար մեծ վտանգ են ներկայացնում, քանի որ շահագործման պրոցեսում մարդը կարող է դիպչել այնպիսի մասերին, որոնք գտնվում են լարման տակ:

Էլեկտրական հոսանքի վտանգավորության պատճառներ են հանդիսանում[a] վնասված հոսանքակիր հաղորդալարերը, ԷՀՄ-ի իրանը և այլ սարքավորումներ, որոնք գտնվում են լարման տակ: Վատ հողանցման, հատակի ոչ ճիշտ ընտրության դեպքում մարդը կարող է ընկնել քայլային լարման տակ, որը հետևանք է վնասված հաղորդալարերի:

Էլեկտրական վնասվածքների կանխարգելման համար կարևոր նշանակություն ունի էլեկտրաանվտանգության ապահովումը:

Հոսանքահարումից պաշտպանվելու համար օգտագործվում են մեկուսիչ յուղեր: Հոսանքահար մասերը փակվում են զանազան մեկուսիչ նյութերով: Սարքերն ու

սարքավորումները հողանցում են: Հոսանքահարվածին անհրաժեշտ է ցուցաբերել առաջին օգնություն, այնուհետև դիմել բժշկի:

3. Հրդեհի ծագման ու տարածման ժամանակ մեծ դեր են խաղում արտադրական շենքերի նախագծական առանձնահատկություններ[a] չափերը, օգտագործված նյութերի տեսակը և այլն:

Ձեռնարկության նախագծման ընթացքում հաշվի է առնվում մարդկանց անվնաս էվակուացման հնարավորությունները: Հնարավորինս կարճ ժամկետում մարդիկ պետք է ազատեն շենքը՝ անցնելով աշխատատեղից մինչև անվտանգ գոտի տանող ճանապարհը: Այն չպետք է անցնի թեք ճանապարհով, պարուրաձև աստիճաններով, խոչընդոտներով:

Հրդեհային անվտանգության կարևոր պայմաններից է հրդեհի առաջին իսկ նշանների դեպքում անմիջապես ահազանգումը հրշեջ ծառայություն: Հրդեհը վերացնելու գործում մեծ դեր ունի հիմնարկության կամավոր հակահրդեհային պաշտպանությունը:

ԷՀՄ-ներով համալրված սենյակներում հրդեհներ կարող են առաջանալ[a] էլեկտրական հաղորդալարերից, էլեկտրական սարքերի խափանումներից, մարդկանց անփույթ վարվելակերպից, ջեռուցման սարքերից (գազ):

4. Արհեստական և բնական լուսավորվածության անբավարարվածությունը մարդկանց մոտ առաջացնում է տեսողական օրգանների գերլարվածություն և հոգնածություն:

Անբավարար լուսավորվածության պատճառ կարող են հանդիսանալ լուսավորման միջոցների ոչ ճիշտ հաշվարկը և տեղադրումը:

Որպես արհեստական լույսի աղբյուր հանդիսանում են շիկացման թելիկով և գազապարպման լամպերը: Լամպերը կոտրվելուց ինչպես նաև լամպի լույսը տվյալ աշխատանքային տեղը ուղղելու և աշխատողին լամպի շլացուցիչ լույսից պաշտպանելու համար օգտագործում են արմատուրա և այդ լամպ արմատուրա միացությունը անվանում են լուսամփոփ:

Բնական լուսավորվածության բացակայությունը կամ անբավարվածությունը աշխատողների մոտ առաջացնում է գերլարվածություն, տեսողական օրգանների գերհոգնածություն:

Բնական լուսավորվածության բացակայությունը կամ անբավարվածությունը առաջանում են պատուհանների բացակայության կամ նրանց չափերի և դիրքի ոչ ճիշտ ընտրության պատճառով:

5. Ճառագայթումից պաշտպանվելու համար օգտագործում են մանիտորների համար նախատեսված հատուկ պաշտպանիչ ֆիլտրեր, որոնք իրենցից ներկայացնում են ապակե էկրաններ որոնք հողանցված են: Ցանկալի է օգտագործել 15” և 17” մանիտորներ քանի որ նրանց էկրաններն իրենց մեջ պարունակում են վերը նշված ֆիլտրերը:

Ճառագայթումը մարդկանց մոտ առաջացնում է տեսողական օրգանների լարվածություն և հոգնածություն: Երկարատև ճառագայթումը կարող է առաջացնել տեսողության թուլացում նույնիսկ նաև կուրացում:

5.2 Բնական լուսավորության կազմակերպումը համակարգչային ստահում

5.2.1 Արտադրական Լուսավորություն

Արտադրական լուսավորության անթերի կազմակերպումն աշխատանքային բարենպաստ պայմանների ստեղծման կարևոր գործոն է:

Թույլ լուսավորությունը ստիպում է մարդուն լարեու տեսողությունը, առաջ է բերում ընդհանուր, և մասնավորապես տեսողական օրգանների հոգնածություն:

Որքան թույլ է լուսավորվածությունը, այնքան ավելի փոքր է աչքի տեսողական սրությունը, այսինքն՝ առավել մանր առարկաները տարբերելու ունակությունը: Թույլ լուսավորվածությունը նվազեցնում է նաև տեսողությամբ ընկալելու արագությունը և աչքի կոնտրաստային զգացողությունը:

Աչքն ընդունակ է հարմարվելու տվյալ լուսավորվածությանը և պայծառությանը:

Հարմարվելու այդ գործողությունը, որը աչքի ադապտացիա է կոչվում, կարող է տևել նույնիսկ մինչև 50 րոպե: Այս ժամանակամիջոցում աչքի տեսողական ունակությունը խիստ նվազում է: Աչքի հաճախակի ադապտացիան ոչ միայն վնասակար է և հոգնեցուցիչ, այլ լարված աշխատանքում ստեղծում է վտանգավոր պահեր և վթարների պատճառ դառնում:

Անհավասարաչափ կամ թույլ լույսի տևական ազդեցությունը կարող է նաև տեսողության մասնիկի կորստի և մի շարք այլ հիվանդությունների պատճառ դառնալ:

Անբարենպաստ լուսավորությունը զգալի չափով նվազեցնում է աշխատանքի արտադրողականությունը, մեծացնում է խոտանի տոկոսը և իջեցնում արտադրանքի որակը:

Լուսավորությունը ունի նաև գեղագիտական նշանակություն: Լավ կազմակերպված լուսավորության պայմաններում աշխատանքը ավելի հաճելի և դյուրին է թվում:

Լուսավորությունը պետք է լինի բավարար և իր սպեկտրով նմանվի բնական լույսին:

Արտադրական շենքում և աշխատատեղերում թանձր ստվերներ կամ կուրացնող պայծառության փայլեր չպետք է լինեն:

Լույսի աղբյուրը, ինչպես և դրանց համար անհրաժեշտ սարքավորումները, պետք է հաճելի տեսք ունենան, լինեն անվնաս և ընտրվեն տնտեսապես նպաստավոր միջոցներով:

5.2.2 Բնական Լուսավորության Առանձնահատկությունները

Բնական լույսը մարդու համար կենսական անհրաժեշտություն է, այն ցրվում է հավասարաչափ և տեսողության համար բարենպաստ է: Հազարամյակների ընդացքում նրան սովորել ու հարմարվել է մարդու աչքը:

Բնական լույսը հանգստացնում է մարդու նյարդային համակարգը, բարձրացնում տրամադրությունն ու աշխատունակությունը, ստեղծում բնության հետ անմիջական կապի զգացողություն:

Այդ պատճառով բնական լուսավորության ճիշտ կազմակերպումը տեխնիկատնտեսական նշանակության հետ միասին ունի նաև հիգիենիկ մեծ նշանակություն:

Բնական լույսով լուսավորվածության աստիճանը խիստ փոփոխական է: Տարվա եղանակի, օրվա ժամերի, ամպամածության, օդի մաքրության և այլ պայմաններից կախված այն փոխվում է:

Արտադրական շենքերի բնական լուսավորվածությունը տրվում է հետևյալ ձևերով՝

Կողքից՝ պատուհաններից կամ թափանցիկ նյութերից պատրաստված պատերից:

Վերևից՝ առաստաղում լուսավորության նպատակով պատրաստված ապակեծածկ մասերից (երդիկից):

Միաժամանակ պատուհաններից և երդիկից (կոմբինացված):

Այս կամ այն ձևի ընտրությունը կախված է շենքի չափերից, արտադրության բնույթից և մի շարք այլ գործոններից:

Քանի որ բնական լույսի ինտենսիվությունը խիստ փոփոխական է, ուստի բնական լույսով շենքերի լուսավորվածության ատիճանը գնահատվում է հարաբերական մեծությամբ՝ բնական լուսավորվածության գործակցով (ԲԼԳ):

Շենքի ներսում, որևէ կետի բնական լուսավորվածության հարաբերությունն է այդ նույն ժամանակ դրսում ամբողջ երկնակամարից ցրված լույսով հորիզոնական հարթության լուսավորությանը՝ արտահայտված տոկոսներով.

$$e = \frac{E_{\acute{Y}}}{E_{1}} * 100\%$$

որտեղ՝ e–ը ներսում, M կետի լուսավորվածության գործակիցն է:

$E_{\acute{Y}}$-ն տվյալ M կետի լուսավորվածությունն է շենքի ներսում,

E_{1}-ն դրսում հորիզոնական հարթության լուսավորվածությունն է բաց երկնակամարից:

Բնական լուսավորվածության գործակցի ամենափոքր հաշվարկային արժեքը որոշվում է դրսում 5000լք լուսավորվածության պայմաններում:

5.2.3 Բնական Լուսավորության Նորմերը

Բնական լուսավորության գործակցի մեծությունը որոշելիս պետք է հաշվի առնել նաև արտադրական շենքի աշխարհագրական տեղը և դիրքը՝

$$e_h = e_{l}mc,$$

որտեղ՝ e_h–ը ԲԼԳ - ի հաշվարկային մեծությունն է,

e–ն ԲԼԳ - ի նորմավորված արժեքն է, որն ընտրվում է վերևի աղյուսակից:

m–ը լուսային կլիմայի գործակիցն է, որի մեծությունը ընտրվում է տեղեկագրքերից և կախված է նրանից, թե որտեղ է գտնվում արտադրական շեմքը:

c –ն կոչվում է արևայնության գործակից: Այն կախված է արեգակի նկատմամբ շենքի գրաված դիրքից և տեղադրման վայրից:

5.2.4 Բնական Լուսավորության Հաշվարկը Աշխատանքային Տեղերում

Աշխատասենյակում, որի երկարությունը 8մ է, լայնությունը՝ 5մ և բարձրությունը՝ 3,2մ հաշվել բնական լուսավորությունը սանիտարական պահանջվող նորմերով ապահովելու համար անհրաժեշտ լուսամուտների ընդհանուր մակերեսը և քանակը:

Նախ պետք է որոշել կատարվող տեսողական աշխատանքի կարգը և բնական լուսավորության գործակիցը այն գոտու համար, որտեղ գտնվում է տվյալ հիմնարկը: Տեսողական աշխատանքը երրորդ կարգի է՝ բարձր ճշտության:

Հայաստանը գտնվում է ընդունված հինգ գոտիներից հինգերորդում, իսկ բնական լուսավորվածության նորման ըստ նորմերի (CH) տրված է երրորդ գոտու համար,այսինքն՝ միջին գոտու հաշվով:

$$ë = \frac{E_l}{E_ŋ} * 100\%; \quad ë = 2\%$$

$$e_h = ë \cdot m \cdot c$$

որտեղ՝ , $ë = 2\%$ m–լուսային կլիմայի գործակիցն է առանց հաշվի առնելու արևային լույսի ուղիղ հոսքը (l-1,2;ll-1,1;lll-1,0;lV-0,9;V-0,8 – ըստ սանիտարական նորմերի),

m=0.8, C- արևայնության գործակից, հաշվի առնելով արևի լույսի ուղիղ հոսքը՝ C=1:

Հետևաբար՝

$$e_h = 2 * 0.8 * 1 = 1.6\%:$$

Բնական լուսավորության հաշվարկը աշխատանքային տեղում կատարվում է սենյակի լուսամուտների ընդհանուր մակերեսի որոշմամբ.

$$S_l = \frac{S_h \cdot e_l \cdot k \cdot \eta_l}{100 \cdot \tau_0 \cdot r_1} \cdot k_{zեկ}$$

որտեղ՝

k- ն պահեստային գործակից (ապակիների կեղտոտվածության և այլն),

ղ - ն լուսամուտների լուսային բնութագիրը, որը որոշվում է ըստ սենյակի պարամետրերի և լուսամուտի վերին մասից մինչև աշխատանքային մակերեսը ունեցած բարձրության հարաբերությամբ՝ a/b և b/hաշխ. A - երկարություն՝ 8մ, b - լայնություն՝ 5մ□(⇒⊥) a/b= 8/5=1,6;

$h_{աշխ}$- հաշվարկային բարձրությունն է, որը $H_{ընդ} - (h_լ + h_մ)$, որտեղ՝ hլ-ն առաստաղից մինչև լուսամուտի վերին մասը ունեցած հեռավորությունն է: Մեր դեպքում hլ=0,3մ; hմ-ն հատակից մինչև աշխատանքային մակերեսը եղած բարձրությունն է, hմ=0,8մ:

Այսպիսով՝

$$H_{աշխ} = H_{ընդ} - (h_լ + h_մ) = 3.2 - (0.3 + 0.8) = 2.1$$

$$\frac{b}{h_{աշխ}} = \frac{5}{2.1} = 2.4$$

Որքան a/b հարաբերությունը փոքր է b/h-ից, այնքան $η_լ$-ն մեծ է(ըստ աղյուսակի մինչև $η$ =66): 1,6/2,4=0,7 այս դեպքի համար $η_լ$=46:

$τ_0$ –լուսաթափանցման ընդհանուր մակերեսի գործակից, ըստ ապակու որակի, շրջանակի տիպի, արևապաշտպան սարքերի և այլն:

$$τ_0 = τ_1 \cdot τ_2 \cdot τ_3 \cdot τ_4 \cdot τ_5$$

որտեղ՝

$τ_1$ -ն ապակու լուսաթափանցման գործակիցն է, սովարական միաշերտ ապակիների դեպքում $τ_1$=0.9,

$τ_2$- գործակից, որը հաշվի է առնում լույսի կորուստները ըստ շրջանակի տիպի. փայտից լինելու դեպքում՝ 0,75 է, մետաղականը՝ 0,6,

$τ_3$- լույսի կորուստները հաշվի առնող գործակից. լինում է 0,5փ 0,8 (ըստ կեղտոտվածության),

$τ_4$ - շինության տանող կոնստրուկցիաներից առաջացող լույսի կորուստները հաշվի առնող գործակից (սյուներ, կողային պատեր և այլն), $τ_4$=1,

$τ_5$ - գործակից. արևապաշտպան սարքերից լույսի կորուստները հաշվի առնող $τ_5$=1:

$$τ_0 = 0.9 \cdot 0.7 \cdot 0.8 \cdot 1 \cdot 1 = 0.5$$

r_1 - գործակից է ըստ a,b,H հարաբերության և դրանց մակերեսի անդրադարձման չափերի. սպիտակեցված առաստաղի դեպքում $p_{առ}$=0.7; սպիտակեցված պատերի դեպքում $p_{պատ}$=0.5; փայտե հատակ՝ չներկված $p_հ$=0.9 և a/b և b/hաշխ հարաբերությունը՝ 0,7-ի: Այս արժեքների դեպքում ըստ սանիտարական նորմերի r_1=6 (r_1լինում է 1փ10):

$k_{շենք}$- գործակից, որը հաշվի է առնում դիմացի շենքի հեռավորությունից գցած ստվերը: Եթե շենքը շատ մոտ չէ $k_{շենք}$=1,7; բացակայության դեպքում $k_{շենք}$=1:

Այսպիսով՝

$$S_l = \frac{40 \cdot 1.6 \cdot 1.5 \cdot 46}{100 \cdot 0.6 \cdot 6} \cdot 1.7 = 20 մ^2; \qquad S_l = 20 մ^2$$

Լուսամուտների քանակը որոշվում է $S_լ/S_հ$ հարաբերությանը համապատասխան՝ ըստ տեսողական աշխատանքի կարգի: Տեսողական աշխատանքը մեր դեպքում բարձր բարձր ճշտության է և այդ հարաբերությունը պետք է բավարարի $\frac{1}{2} \div \frac{1}{5}$-ին:

$$\frac{S_լ}{S_հ} = \frac{20}{40} = \frac{1}{2}$$

Այս հարաբերությունը բավարար է ապահովելու համար անհրաժեշտ լույսի քանակը բարձր ճշտության աշխատանք կատարելու համար:

Գլուխ 6: Բնապահպանության հարցերի հիմնավորում

Գլուխ 6.1 Արհեստական աղբյուրների առաջացրած էլեկտրամագնիսական դաշտերը և դրանց ազդեցությունը շրջակա միջավայրի վրա

Էվոլյուցիայի և կենսագործունեության գործընթացում մարդ ենթարկվում է բնական էլեկտրամագնիսական ֆոնի ազդեցությանը, որի բնութագրերը օգտագործվում են որպես ինֆորմացիայի աղբյուր՝ ապահովելով անընդհատ փոխազդեցությունը արտաքին միջավայրի փոփոխվող պայմանների հետ: Ժամանակակից հետազոտությունների արդյունքերը վկայում են այն մասին, որ բոլոր կենդանի օրգանիզմները, միաբջջից մինչև բարձրագույն կենդանիներ և մարդ, դրսևորում են բացառապես բարձր զգայունություն էլեկտրական և մագնիսական դաշտերի նկատմամբ, որոնց պարամետրերը մոտ են կենսոլորտի դաշտերի բնական

պարամետրերին: Բազմաթիվ վիճակագրական տվյալներով ցույց է տրված, որ բնական աղբյուրների (գեոմագնիսական դաշտեր, մթնոլորտային լեցքեր, ասղերի և գալակտիկայի ճառագայթում) էլեկրտրամագնիսական դաշտերը էապես ազդում են կենսաբանական ռիթմերի ձևավորման վրա: Հայտնաբերվել են բավականին ստույգ փոխադարձ կապեր արևային և գեոմագնիսական ակտիվության և հիպերտոնիկ կրիզի թվի աճի, սրտամկանի ինֆարկտի, հոգեկան խանգարումենրի միջև:

Վերջին ժամանակներս մարդու և էլեկտրամագնիսական դաշտերի փոխազդեցության խնդիրը դարձել է հրատապ՝ կապված ռադիոկապի և ռադիոլոկացիայի ինտենսիվ զարգացման, տեխնոլոգիական գործընթացների իրականացման համար էլեկտրամագնիսական էներգիայի կիրառության ոլորտի ընդլայնման, կենցաղային էլէկտրական և ռադիոէլեկտրոնային սարքերի մասսայական տարածման հետ:

Եթե դեռ 20-25 տարի առաջ էլեկտրամագնիսական ճառագայթումից պաշտպանվելու խնդիրը վերաբերվում էր հիմնականում արտադրական պայմաններին (ռադիոլոկացիոն կայանների աշխատակազմ, տեխնոլոգիական սարքավորումների օպերատորներ), ապա այսօր բնակչության մեծամասնությունը, փաստորեն, ապրում է արհեստական բնույթի էլեկտրամագնիսական դաշտերում, որոնք օժտված են բավականին բարդ տարածական կառուցվածքով: Արհեստական աղբյուրները ստեղծում են ավելի մեծ ինտենսիվության էլեկտրամագնիսական դաշտեր, քան բնական աղբյուրները:

Կլինիկական հետազոտություններով հաստատված է, որ արհեստական ծագումով էլեկտրամագնիսական դաշտերը խաղում են որոշակի դեր սրտանոթային, ուռուցքային, ալերգիկ և արյան հիվանդությունների զարգացման գործում, ինչպես նաև կարող են ազդել գենետիկ կառուցվածքի վրա: Սիստեմատիկ ազդեցության պայմաններում էլեկտրամագնիսական դաշտերը առաջացնում են բնակչության առողջական վիճակի արտահայտված փոփոխություններ, այդ թվում էլեկտրամագնիսական դաշտերի աղբյուրների հետ մասնագիտորեն կապ չունեցող անձանց վրա, ընդ որում թույլ ինտենսիվության դաշտերի ազդեցության էֆեկտը կրում է հեռակա բնույթ:

Մարդը միայն հատուկ սարքերի միջոցով կարող է որոշել էլեկտրական լարման առկայությունը կամ բացակայությունը: Դա է տարբերությունը այլ վնասակար գործոններից: Հետևապես, օրգանիզմի պաշտպանական ռեակցիան ի հայտ է գալիս էլեկտրական հոսանքի ազդեցության հոսանքի ազդեցության տակ մարդու ընկնելուց հետո:

Էլեկտրական հոսանքի, էլեկտրական աղեղի, էլեկտրամագնիսական դաշտի, ստատիկ էլեկտրական հոսանքի ազդեցությունները առաջացնում են էլեկտրական վնասվածքներ: Պայմանականորեն այս վնասվածքները կարելի է բաժանել 3 խմբի՝ տեղային վնասվածք, ընդհանուր կամ էլեկտրական հարվածներ և խառը վնասվածք:

Հոսանքը, որն անցնում է մարդու մարմնով և հպման լարումը արտահայվում են հետևյալ բանաձևերով.

$$I = UG_h \frac{Y_B(1-a^2) + Y_C(1-c)}{Y_A + Y_B + Y_C + G_h}$$

$$U_h = U \frac{Y_B(1-a^2) + Y_C(1-c)}{Y_A + Y_B + Y_C + G_h}$$

Որտեղ Y_A, Y_B, Y_C – մեկուսացված, ֆազային հաղորդալարերի կոմպլեքս լրիվ հաղորդականություններն են:

$$Y_A = \frac{1}{R_A} + j\omega C_A$$

$$Y_B = \frac{1}{R_B} + j\omega C_B$$

$$Y_C = \frac{1}{R_C} + j\omega C_C$$

որտեղ U-ն ցանցի ֆազային լարումն է (Վ),

$G_h = \frac{1}{R_h}$ մարդու մարմնի հաղորդականությունն է,

a-ն ֆազային գործակից, հաշվի է առնում ֆազերի միջև տեղափոխությունը:

Ֆազային հաղորդալարերի հաղորդականության հավասարության դեպքում $Y_A = Y_B = Y_C = Y$`

Հոսանքը, որը անցնում է մարդու, կորոշվի`

$I_h = UG_h \frac{2Y}{3Y + G_h}$ կամ

$$I_h = \frac{U}{R_h + Z/3}$$

Որտեղ Z-ը ֆազային հաղորդալարի լրիվ դիմադրությունն է (կապված հողի հետ) կոմպլեքս տեսքիով (ՕՄ):

$$Z = \frac{1}{Y} = \frac{1}{\frac{1}{R} + j\omega C}$$

Որտեղ R-ը ակտիվ, մեկուսացված ֆազային հաղորդալարի դիմադրությունն է,

C-ն մեկուսացված ֆազային հաղորդալարի ունակությունն է՝ հողի հետ կապված:

Հողի հետ կապված մեկուսացված ֆազային հաղորդալարերի դիմադրությունների հավասարության դեպքում ($R_A = R_B = R_C = R$) և ունակությունների բացակայության դեպքում ($C_A = C_B = C_C = C = 0$).

$$I_h = \frac{U}{R_h + R/3}$$

Ցանցի վթարային աշխատանքային ռեժիմում, երբ հաղորդալարերից մեկը հողի հետ շղթա է կազմել, մարդու մարմնով անցնող հոսանքը կլինի՝

$$I_h = \frac{U\sqrt{3}}{R_h + R_\psi}$$

Քանի որ հաճախակի տեղի ունի $R_0 << R_\psi$ պայմանը, ապա

$$I_h = \frac{U\sqrt{3}}{R_h}$$

$$U_h = U\sqrt{3}$$

Եզրակացություն

Մոբայլ աշխարհի զարգացման տեմպերը գերազանցում են ամեն տեսակի կանխատեսումները: Այսօր ավելի հավանական է հանդիպել մարդու ով իր հետ չի վերցրել անձնագիրը, քան մեկին, ով առանց մոբայլ սարքի է դուրս եկել: Այսպես 21-րդ հազարամյակը աստիճանաբար մեր կյանք մտցրեց սմարթֆոնները: Ստատիկ համակարգիչների համար ծրագրային ապահովում ստեղծող գիգանտները, ստեղծեցին մոբայլ օպերացիոն համակարգեր: Այդպես Մայքրոսոֆթ ընկերությունը առաջադրեց Windows Phone օպերացիոն համակարգը: Այն այժմ երրորդն է շուկայում, ըստ օգտագործողների թվի: Համակարգը, ի տարբերություն մրցակից Android օպերացիոն համակարգից, սմարթֆոնի միջոցով ֆայլային համակարգը ուղիղ կառավարելու միջոցներ չի ընձեռում: Օգտագործողները իրենց հերթին առօրյայում հաճախակի սեփական կամքով կամ քաղաքավարությունից դրդված ստիպված են լինում տրամադրել իրենց սմարթֆոնները այլ մարդկանց: Այդ պատճառով առաջանում է պահանջարկ օգտագործողի որոշակի անձնական օգտագործման տվյալներ թաքուն պահել: Հետևյալ աշխատանքի շրջանակներում մշակվեց Windows Phone 8 օպերացիոն համակարգի համար նախատեսված հավելված, որը ունենալով երկու՝ կեղծ և իրական, ինտերֆեյսներ, դաշնամուր է անիրավասու անձի համար և ֆայլերի թաքցման և դիտարկման ծրագիր իրավասու անձի՝ տվյալ դեպքում սմարթֆոնի տիրոջ համար:

Համակարգը կարող է լայնորեն կիրառվել անհատների, խմբաավորումների և գաղտնի ծառայությունների կողմից, պայմանով, որ վերջիններս չեն բացահայտի համակարգի գոյության փաստը: Հաշվի առնելով, որ համակարգը պաշտպանելու է իր լիցենզիան գնողներին, կարելի է ենթադրել, որ վերջիններս սեփական անվտանգությունը բարձր պահհելու նպատակով, չեն խախտի լիցենզիոն համաձայնագիրը:

Գրականություն

1. Эндрю Троелсен - Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание, 2013
2. Shawn Wildermuth - Essential Windows Phone 8 (2nd Edition)
3. Whitechapel A., McKenna S. - Windows Phone 8 Development Internals, 2013
4. Falafel Software - Pro Windows Phone App Development, 2013
5. Rahman M. - Expert C# 5.0: with the .NET 4.5 Framework, 2012
6. Lori A. MacVittie - XAML in a Nutshell, 2006
7. Arash Habibi Lashkari - Mobile Operating Systems and Programming: Mobile Communications, 2011
8. Daniel Vaughan - Windows Phone 8 Unleashed, 2013

Հավելված

FileManager.cs

```
using Microsoft.Phone.Shell;
using Microsoft.Xna.Framework.Media;
using PianoPhone.IO;
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.IO.IsolatedStorage;
using System.Linq;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Json;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using Windows.Phone.PersonalInformation;
using Windows.Storage.Streams;

namespace PianoPhone
{
   class FileManager
   {

      static Task<IsolatedStorageFileStream> OpenFileAsync(string path, FileMode fileMode, FileAccess fileAccess )
      {
         return Task.Run(() =>
         {
            using (IsolatedStorageFile file = IsolatedStorageFile.GetUserStoreForApplication())
            {
               return file.OpenFile(path, fileMode, fileAccess);
            }
         });
      }

      static Task SaveFileAsync(string path, Stream data)
      {
         return Task.Run(async () =>
         {
            using (IsolatedStorageFile file = IsolatedStorageFile.GetUserStoreForApplication())
            {
               using (IsolatedStorageFileStream fs = await OpenFileAsync(path, FileMode.OpenOrCreate, FileAccess.Write))
               {
                  byte [] buffer = new byte[data.Length];
                  data.Seek(0, SeekOrigin.Begin);
                  await data.ReadAsync(buffer,0,buffer.Length);
                  await fs.WriteAsync(buffer, 0, buffer.Length);
```

```
            }
        }

    });
}

static Task SerializeAndSaveFile<T>(T source, string path)
{
    return Task.Run(async () =>
        {

            DataContractJsonSerializer jserializer = new DataContractJsonSerializer(typeof(T));
            Stream destination = new MemoryStream();
            Stream tempStream = new MemoryStream();
            jserializer.WriteObject(tempStream, source);

            //Write length
            int jsonPartSize = (int)tempStream.Position;
            byte [] jsonLengthArray = BitConverter.GetBytes(jsonPartSize);
            await destination.WriteAsync(jsonLengthArray, 0, jsonLengthArray.Length);
            tempStream.Seek(0, SeekOrigin.Begin);

            //Write File
            await tempStream.CopyToAsync(destination);

            //Write image

            if ((source as IData).Data != null)
            {
                (source as PFile).Data.Seek(0, SeekOrigin.Begin);
                await (source as IData).Data.CopyToAsync(destination);
            }

            return SaveFileAsync(path, destination);
        });
}

async static Task<T> DeserializeAndOpenAsync<T>(string path, FileMode mode, FileAccess access) where T: class
{
    using (Stream source = await OpenFileAsync(path, mode, access))
    {
        byte[] lengthArray = new byte[sizeof(int)];
        await source.ReadAsync(lengthArray, 0, lengthArray.Length);
        int jsonLength = BitConverter.ToInt32(lengthArray, 0);

        byte[] jsonArray = new byte[jsonLength];
        await source.ReadAsync(jsonArray, 0, jsonArray.Length);
        DataContractJsonSerializer jserializer = new DataContractJsonSerializer(typeof(T));
        Stream jsonStream = new MemoryStream();
        await jsonStream.WriteAsync(jsonArray, 0, jsonArray.Length);
        jsonStream.Seek(0, SeekOrigin.Begin);
        IData result = jserializer.ReadObject(jsonStream) as IData;
        byte[] imageBuffer = new byte[source.Length - source.Position];
        await source.ReadAsync(imageBuffer, 0, imageBuffer.Length);

        result.Data = new MemoryStream();
        await result.Data.WriteAsync(imageBuffer, 0, imageBuffer.Length);
        result.Data.Seek(0, SeekOrigin.Begin);
        return result as T;
    }
}
```

```
        public static Task<List<PPhoto>> GetPhotosAsync(string albumPath, CancellationToken token)
        {
            return  Task.Run(async () =>
                {
                    List<PPhoto> photos = new List<PPhoto>();
                    using (IsolatedStorageFile store = IsolatedStorageFile.GetUserStoreForApplication())
                    {
                        try
                        {
                            string[] fileNames = store.GetFileNames(Path.Combine(albumPath, "*"));
                            foreach (var fileName in fileNames)
                            {
                                string filePath = Path.Combine(albumPath, fileName);
                                photos.Add(await DeserializeAndOpenAsync<PPhoto>(filePath, FileMode.Open,
FileAccess.ReadWrite));
                            }
                        }
                        catch
                        {
                            return photos;
                        }
                        return photos;
                    }
                });
        }

        public static Task<List<PFile>> GetPhotoAlbums(CancellationToken token)
        {
            return Task.Run(async () =>
                {
                    List<PFile> albums = new List<PFile>();
                    try
                    {
                        using (IsolatedStorageFile store = IsolatedStorageFile.GetUserStoreForApplication())
                        {
                            var fileNames = store.GetFileNames(Directories.Thumbnails + "\\*");
                            foreach (var fileName in fileNames)
                            {
                                string filePath = Path.Combine(Directories.Thumbnails, fileName);
                                albums.Add( await DeserializeAndOpenAsync<PFile>(filePath, FileMode.Open, FileAccess.ReadWrite));
                            }
                        }
                        return albums;
                    }
                    catch
                    {
                        return albums;
                    }
                }, token);
        }

        public static Task<List<PContact>> GetContactsAsync(CancellationToken token)
        {
            return Task.Run(async () =>
            {
                List<PContact> contacts = new List<PContact>();
                try
                {
                    using (IsolatedStorageFile store = IsolatedStorageFile.GetUserStoreForApplication())
                    {
```

```
                token.ThrowIfCancellationRequested();
                var fileNames = store.GetFileNames(Directories.Contacts + "\\*");
                foreach (var fileName in fileNames)
                {
                    token.ThrowIfCancellationRequested();
                    string filePath = Path.Combine(Directories.Contacts, fileName);

                    token.ThrowIfCancellationRequested();
                    PContact contact =  await DeserializeAndOpenAsync<PContact>(filePath, FileMode.Open,
FileAccess.ReadWrite);
                    token.ThrowIfCancellationRequested();
                    Stream dataStr = contact.Data;
                    contact.Data = new MemoryStream();
                    await dataStr.CopyToAsync(contact.Data);
                    contact.Data.Seek(0, SeekOrigin.Begin);
                    contacts.Add(contact);
                }
            }
            token.ThrowIfCancellationRequested();
            return contacts;
        }
        catch
        {
            return contacts;
        }
    }, token);
 }
 public static IEnumerable<string> GetPhotoAlbumNames()
 {
    List<string> directories = new List<string>();
    using (IsolatedStorageFile file = IsolatedStorageFile.GetUserStoreForApplication())
    {
       return file.GetDirectoryNames(Path.Combine(Directories.AlbumRoot, "*"));
    }
 }

 static Task<bool> CreatePhotoAlbumAsync(string albumName, Stream thumbPicture)
 {
    return Task.Run(async () =>
       {
          using (IsolatedStorageFile file = IsolatedStorageFile.GetUserStoreForApplication())
          {
             if (file.DirectoryExists(Path.Combine(Directories.AlbumRoot, albumName)))
             {
                thumbPicture.Dispose();
                return false;
             }
             file.CreateDirectory(Path.Combine(Directories.AlbumRoot, albumName));
             PFile album = new PFile();
             album.Name = albumName;
             album.Path = Path.Combine(Directories.Thumbnails, albumName);
             using(thumbPicture)
             {
                byte [] buffer=  new byte[thumbPicture.Length];
                thumbPicture.Seek(0, SeekOrigin.Begin);
                await thumbPicture.ReadAsync(buffer,0, buffer.Length);
                album.Data = new MemoryStream();
               await album.Data.WriteAsync(buffer, 0, buffer.Length);
             }
             await SerializeAndSaveFile<PFile>(album,album.Path);
             return true;
```

```
            }
        });
    }

    static bool PhotoAlbumExcists(string albumName)
    {
        using (IsolatedStorageFile store = IsolatedStorageFile.GetUserStoreForApplication())
        {
            return store.DirectoryExists(Path.Combine(Directories.AlbumRoot, albumName));
        }
    }

    public static async Task<int> AddPhotosAsync(IEnumerable<PPhoto> photos, CancellationToken token)
    {
        return await Task.Run(async () =>{
            int count = 0;
        foreach (var photo in photos)
        {
            if (!PhotoAlbumExcists(photo.AlbumName))
            {
                await CreatePhotoAlbumAsync(photo.AlbumName, photo.Data);
                token.ThrowIfCancellationRequested();
            }
            await SerializeAndSaveFile<PPhoto>(photo, photo.Path);
            count++;
            token.ThrowIfCancellationRequested();
        }
        return count;
        },token);
    }

    public static void CreateDefaultDirectories()
    {
        using (IsolatedStorageFile file = IsolatedStorageFile.GetUserStoreForApplication())
        {
            if (file.DirectoryExists(Directories.Root))
            {
                return;
            }
            else
            {
                file.CreateDirectory(Directories.Root);
                file.CreateDirectory(Directories.AlbumRoot);
                file.CreateDirectory(Directories.DefaultAlbum);
                file.CreateDirectory(Directories.Contacts);
                file.CreateDirectory(Directories.Thumbnails);
            }
        }
    }


public async static Task SaveItemsAsync(List<Models.CollectionControlModel> list)
    {
        var data=  list.First().Data;
        if (data is PictureAlbum)
        {
            foreach (var album in list)
            {
                var pictureAlbum = album.Data as PictureAlbum;
                if (await CreatePhotoAlbumAsync(pictureAlbum.Name, pictureAlbum.Pictures.First().GetThumbnail()))
                {
```

```
            List<PPhoto> photos = new List<PPhoto>();
            foreach (var picture in pictureAlbum.Pictures)
            {
                PPhoto photo = new PPhoto();
                photo.AlbumName = pictureAlbum.Name;
                photo.Name = picture.Name;
                photo.Path = Path.Combine(Directories.AlbumRoot, photo.AlbumName, photo.Name);
                await Task.Run(async () =>
                {
                    using (var source = picture.GetImage())
                    {
                        byte[] arr = new byte[source.Length];
                        await source.ReadAsync(arr, 0, arr.Length);
                        photo.Data = new MemoryStream();
                        await photo.Data.WriteAsync(arr, 0, arr.Length);
                    }
                });
                photos.Add(photo);
            }
            await FileManager.AddPhotosAsync(photos, CancellationToken.None);
        }
        else
        {
            ShellToast toast = new ShellToast();
            toast.Content = "Album is already imported";
        }
    }
}
else
{
    if (data is Picture)
    {
        string albumName = (data as Picture).Album.Name;
        if (!PhotoAlbumExcists(albumName))
        {
            await CreatePhotoAlbumAsync(albumName,(data as Picture).GetThumbnail());
        }
        List<PPhoto> photos = new List<PPhoto>();
        foreach (var p in list)
        {
            Picture picture = p.Data as Picture;
            PPhoto photo = new PPhoto();
            photo.AlbumName = albumName;
            photo.Name = picture.Name;
            photo.Path = Path.Combine(Directories.AlbumRoot, photo.AlbumName, photo.Name);
            await Task.Run(async () =>
            {
                using (var source = picture.GetImage())
                {
                    byte[] arr = new byte[source.Length];
                    await source.ReadAsync(arr, 0, arr.Length);
                    photo.Data = new MemoryStream();
                    await photo.Data.WriteAsync(arr, 0, arr.Length);
                }
            });
            photos.Add(photo);
        }
        await FileManager.AddPhotosAsync(photos, CancellationToken.None);
    }
    else
    {
```

```
            if (data is PContact)
            {
                List<PContact> contacts = new List<PContact>();
                foreach (var c in list)
                {
                    contacts.Add( c.Data as PContact);
                }
                FileManager.AddContactsAsync(contacts, CancellationToken.None);
            }
        }
    }
}

async static void AddContactsAsync(IEnumerable<PContact> contacts, CancellationToken cancellationToken)
{
   await Task.Run(async () =>
    {
        int count = 0;
        foreach (var contact in contacts)
        {
            await SerializeAndSaveFile<PContact>(contact, contact.Path);
            count++;
            cancellationToken.ThrowIfCancellationRequested();
        }
        return count;
    }, cancellationToken);
}

public static Task<List<SerializableContact>> GetContactsFromPhoneAsync(CancellationToken token)
{
    TaskCompletionSource<List<SerializableContact>> tcs = new TaskCompletionSource<List<SerializableContact>>();
    Microsoft.Phone.UserData.Contacts contactsAdaptor = new Microsoft.Phone.UserData.Contacts();
    token.ThrowIfCancellationRequested();
    contactsAdaptor.SearchCompleted += async (s, e) =>
    {
        token.ThrowIfCancellationRequested();
        List<SerializableContact> contacts = new List<SerializableContact>();
        var result = e.Results;
        token.ThrowIfCancellationRequested();
        foreach (var contact in result)
        {
            SerializableContact sContact = new SerializableContact(contact);
            await sContact.InitializeData(contact);
            contacts.Add(sContact);
            token.ThrowIfCancellationRequested();
        }
        try
        {
            tcs.SetResult(contacts);
        }
        catch (OperationCanceledException)
        {
            tcs.TrySetCanceled();
        }
        catch (Exception ex)
        {
            tcs.TrySetException(ex);
        }
    };
    contactsAdaptor.SearchAsync(String.Empty, Microsoft.Phone.UserData.FilterKind.None, "Contacts Operation");
    return tcs.Task;
```

```
    }
  }
}
```

PianoEngine.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Coding4Fun.Toolkit.Audio;
using Coding4Fun.Toolkit.Audio.Helpers;
using System.IO;
using Microsoft.Xna.Framework.Audio;
using System.Threading;

namespace PianoPhone
{
  class PianoEngine
  {
     const int KeyCount = 12;
    List<MemoryStream> samples;
    List<SoundEffect> soundEffects;
    List<SoundEffectInstance> soundEffectInstances;

    public PianoEngine()
    {
      InitSoundEffects();

    }

    private void InitSamples()
    {
      samples = new List<MemoryStream>();
      for(int i = 0; i < KeyCount; i++)
      {
        string address = "Assets/Piano Keys/Piano" + i.ToString();
        using (FileStream fStream = new FileStream(address, FileMode.Open))
        {
          using (MemoryStream ms = new MemoryStream())
          {
            byte[] buffer = new byte[fStream.Length];
            fStream.ReadAsync(buffer,0, buffer.Length);
            ms.WriteAsync(buffer,0,buffer.Length);
            ms.Seek(0, SeekOrigin.Begin);
            ms.Close();
            samples.Add(ms);
          }
          fStream.Close();
        }
      }
    }

    private async void InitSoundEffects()
    {
      soundEffects = new List<SoundEffect>();
      soundEffectInstances = new List<SoundEffectInstance>();
      for (int i = 1; i <= KeyCount; i++)
      {
```

```
            await ReadFile(i);
            Thread.Sleep(100);
        }
    }

    private async Task ReadFile(int i)
    {
        string address = "Assets/Piano Keys/Piano" + i.ToString() + ".wav";
        FileStream fStream = new FileStream(address, FileMode.Open);
        byte[] buffer = new byte[fStream.Length + 4 - fStream.Length % 4];
       await fStream.ReadAsync(buffer, 0, buffer.Length);

        soundEffects.Add(new SoundEffect(buffer, 44100, AudioChannels.Stereo));
        fStream.Close();
        fStream.Dispose();
        soundEffectInstances.Add(soundEffects.Last().CreateInstance());
    }

    async public void PlaySample(int i)
    {
        foreach (var soundEffect in soundEffectInstances)
        {
            if (soundEffect.State == SoundState.Playing)
                soundEffect.Stop();
        }
        i--;
        if (soundEffectInstances[i].State == SoundState.Playing)
            soundEffectInstances[i].Stop();
        await Task.Delay(20);
        soundEffectInstances[i].Play();
    }

  }
}
```

PFile.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Serialization;
using System.Text;
using System.Threading.Tasks;

namespace PianoPhone.IO
{
  [DataContract]
  public class PFile:IData
  {

    [DataMember]
    public string Path { get; set; }

    [DataMember]
    public string Name { get; set; }

    public MemoryStream Data
    {
        get;
```

```
            set;
        }
    }

    public interface IData
    {
        MemoryStream Data { get; set; }
    }
}
```

PContact.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.Text;
using System.Threading.Tasks;
using Windows.Phone.PersonalInformation;

namespace PianoPhone.IO
{
    [DataContract]
    public class PContact:PFile
    {
        [DataMember]
        public SerializableContact Contact { get; set; }
    }
}
```

PPhoto.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.Text;
using System.Threading.Tasks;

namespace PianoPhone.IO
{
    [DataContract]
    class PPhoto: PFile
    {
        [DataMember]
        public string AlbumName { get; set; }
    }
}
```

CollectionControlModel.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using System.Windows.Media.Imaging;

namespace PianoPhone.Models
{
   public class CollectionControlModel:INotifyPropertyChanged
   {
      string fileName;
      public string FileName
      {
         get { return fileName; }
         set { fileName= value;
         OnPropertyChanged("FileName");
         }
      }

      BitmapSource thumbnail;
      public BitmapSource Thumbnail
      {
         get { return thumbnail; }
         set
         {
            thumbnail = value;
         OnPropertyChanged("Thumbnail");
         }
      }

      public object Data { get; set; }
      public event PropertyChangedEventHandler PropertyChanged;
      private void OnPropertyChanged(string p)
      {
         if (PropertyChanged != null)
         {
            PropertyChanged(this, new PropertyChangedEventArgs(p));
         }
      }
   }
}
```

CollectionControl.xaml

```
<UserControl x:Class="PianoPhone.CollectionControl"
          x:Name="collectionControl"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   xmlns:controls="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
   xmlns:toolkit="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls.Toolkit"
   mc:Ignorable="d"
   FontFamily="{StaticResource PhoneFontFamilyNormal}"
   FontSize="{StaticResource PhoneFontSizeNormal}"
   Foreground="{StaticResource PhoneForegroundBrush}"
   d:DesignHeight="480" d:DesignWidth="480">
   <UserControl.Resources>
      <DataTemplate x:Key="CollectionControlCellItemTemplate">
         <Grid Tap="Grid_Tap_1" Margin="6" >
            <Grid.RowDefinitions>
               <RowDefinition Height="*"/>
               <RowDefinition Height="auto"/>
            </Grid.RowDefinitions>
```

```
            <Image CacheMode="BitmapCache" Source="{Binding Thumbnail}"
                 HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
                 Stretch="UniformToFill" Width="200" Height="200"/>
            <TextBlock Text="{Binding FileName}" Grid.Row="0" VerticalAlignment="Bottom"
                  Width="200" HorizontalAlignment="Left" TextWrapping="Wrap"/>
          </Grid>
        </DataTemplate>
        <DataTemplate x:Key="CollectionControlListModeItemTemplate">
          <Grid Tap="Grid_Tap_1">
          <StackPanel  Orientation="Horizontal" Height="72">
            <Image Height="60" Width="60" Stretch="UniformToFill"
                 CacheMode="BitmapCache" Source="{Binding Thumbnail}"
                 HorizontalAlignment="Center" VerticalAlignment="Stretch"/>
            <TextBlock Height="60" Margin="24,6" Text="{Binding FileName}" Grid.Row="1" VerticalAlignment="Center"
                  FontSize="24" HorizontalAlignment="Left" TextWrapping="Wrap"/>
          </StackPanel>
            </Grid>
        </DataTemplate>
    </UserControl.Resources>
    <Grid x:Name="LayoutRoot" DataContext="{Binding}" Background="{StaticResource PhoneChromeBrush}">
      <toolkit:LongListMultiSelector Name="longListSelector"  ItemTemplate="{StaticResource
CollectionControlCellItemTemplate}"
                    DataContext="{Binding}" GridCellSize="200,200"
                      ItemsSource="{Binding Items}"
                        SelectionChanged="longListSelector_SelectionChanged_1"
                     />
    </Grid>
</UserControl>
```

TaskExtension.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PianoPhone
{
  public static class TaskExtension
   {
      public static Task ExecuteAndGetTask(Action f)
      {
         return ExecuteAndGetTask<object>(() =>
         {
            f.SafeInvoke();
            return null;
         });
      }

      public static Task<T> ExecuteAndGetTask<T>(Func<T> f)
      {
         if (f != null)
         {
            var ts = new TaskCompletionSource<T>();
            try
            {
               ts.TrySetResult(f());
            }
            catch (OperationCanceledException)
            {
```

```
                ts.TrySetCanceled();
            }
            catch (Exception e)
            {
                ts.TrySetException(e);
            }
            return ts.Task;
        }
        return Task.FromResult<T>(default(T));
    }

    public static void SafeInvoke(this Action action)
    {
        if (action != null)
        {
            action();
        }
    }
  }
}
```